Guía de Referencia del

COMMODORE 64



commodore
COMPUTER

microelectrónica
y control, s.a
µEC

EN TORNO A LA GUIA DE REFERENCIA DEL PROGRAMADOR DEL COMMODORE 64

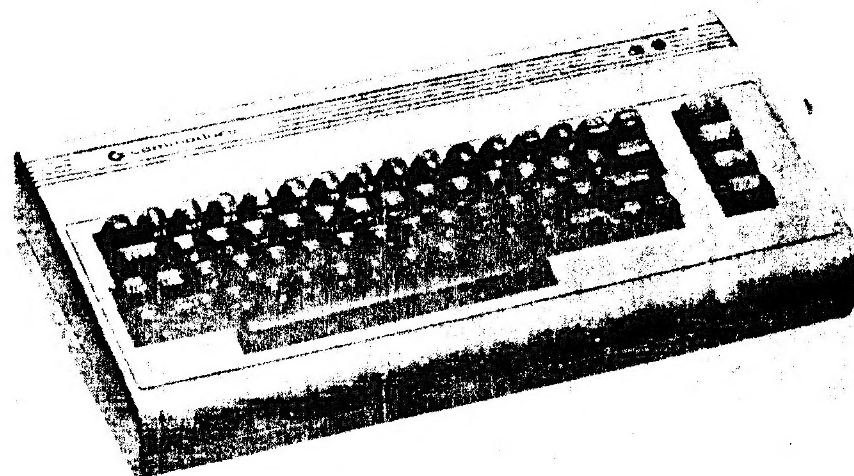
Juegos en cartucho... Sonido espectacular... Gráficos de gran efecto... y una gran capacidad a un precio ajustado hacen del **COMMODORE 64** el mejor **ORDENADOR PERSONAL** en su clase para aplicaciones domésticas, de gestión y para uso como auxiliar en la Educación. La **GUIA DE REFERENCIA DEL PROGRAMADOR DEL COMMODORE 64** explica todo lo que Ud. necesita saber acerca de su **COMMODORE 64**. Es el complemento perfecto del **MANUAL DEL USUARIO,** conteniendo información detallada tanto de las posibilidades gráficas y de sonido como de la utilización de rutinas en Codigo-Máquina. Es el libro de mayor utilidad para un principiante como para un programador experto.
Para el principiante las características mas complejas se explican con la ayuda de programas de ejemplo y un estilo de redacción fácil de leer. Para el programador experto, este libro se ha revisado a conciencia teniendo en cuenta sus necesidades. Y se ha escrito para que cualquier usuario pueda obtener de su **COMMODORE 64** el máximo de sus grandes posibilidades.

commodore

Distribuidor en exclusiva para España
microelectrónica y control, s.a. MEC

Valencia, 49-53 - Teléfono 325 50 08 * - Telex 50916 myc b - 08015 BARCELONA
Ardemans, 8 - 28028 MADRID

Guía de Referencia del COMMODORE 64



commodore COMPUTER

microelectrónica y control, s.a. MEC

Este Manual ha sido traducido por **Jaume Juliá**

La edición de este manual ha sido realizada por el proceso TYPECRAFT-ITC-FOTOTIPO por conversión directa del diskette (floppy) a Fotocomposición, a través de los sistemas COMMODORE.

Dep. Legal: B. 33182-1983

ES UNA REALIZACION ITC Tel. (93) 230 98 04 Barcelona España

CONTENIDO

INTRODUCCION

El MANUAL DE REFERENCIA DEL PROGRAMADOR ha sido desarrollado como una herramienta de trabajo y fuente de consulta para que pueda aprovechar al máximo las posibilidades que le ofrece su COMMODORE 64. Este manual contiene la información que necesita para sus programas, desde el más simple ejemplo hasta la programación más compleja. El MANUAL DE REFERENCIA DEL PROGRAMADOR está diseñado para todos los niveles. Desde el programador principiante hasta el profesional experto en el código máquina del 6502 encontrarán información para desarrollar sus programas. Al mismo tiempo este libro le muestra la potencia de su COMMODORE 64.

Este manual no ha sido diseñado para enseñar el lenguaje de programación BASIC o el lenguaje máquina del 6502. Hay, sin embargo, un extenso glosario de términos, y muchas secciones del libro tienen la clara intención de que aprenda fácilmente. Si usted no tiene conocimientos del lenguaje BASIC y de su uso en programas le sugerimos que estudie el MANUAL DEL USUARIO DEL COMMODORE 64 que acompaña a su ordenador. El MANUAL DEL USUARIO le proporciona una fácil y amena introducción al lenguaje BASIC. Si encuentra dificultades en comprender el funcionamiento del BASIC le remitimos al apéndice F (o al apéndice N en el MANUAL DEL USUARIO) donde encontrará extensa bibliografía sobre el tema.

El MANUAL DE REFERENCIA DEL PROGRAMADOR es justo eso: una referencia. Como la mayoría de libros de referencia, su habilidad en aplicar la información creativamente depende realmente de sus conocimientos. En otras palabras, si usted es un programador principiante no podrá aprovechar al máximo la información contenida en el libro hasta que no amplíe sus conocimientos.

Lo que se pretende en este libro es que encuentre una considerable cantidad de información de referencia para la programación escrita en un estilo claro, explicando la "jerga" utilizada por los programadores. En el lado opuesto, el programador profesional encontrará toda la información necesaria para usar con efectividad todas las posibilidades del COMMODORE 64.

¿QUE SE INCLUYE EN EL MANUAL?

- Nuestro "diccionario BASIC" incluye todos los comandos del BASIC de Commodore, instrucciones y funciones en orden alfabético. Hemos creado una lista que contiene todas las palabras y sus abreviaciones, seguida de una sección donde se explica con detalle el uso de cada palabra, ilustrándolo con pequeños programas que muestran cómo trabaja.

gos de vídeo, crear dibujos animados, componer melodías y muchas cosas más. Si posee el COMMODORE 64 para realizar una sola de las tareas descritas más abajo, ha aprovechado el dinero que pagó por él. Pero el 64 es un completo ordenador que puede realizar ¡CUALQUIER cosa de la lista y alguna más! Además, usted puede conseguir ideas prácticas y creativas afiliándose al Club de Usuarios Commodore de su ciudad y suscribiéndose a la revista mensual CLUB COMMODORE, publicada en España por Microelectrónica y Control.

APLICACION	**COMENTARIOS/REQUISITOS**
JUEGOS DE ACCION	Encontrará los más populares juegos de vídeo como Omega Race, Gorf y Wizard of War, y buenos juegos para aprender como Profesor de matemáticas I, Babysitter hogareña y Artista con Commodore.
PUBLICIDAD	Conecte el COMMODORE 64 a un televisor, póngalo en el escaparate de su tienda y utilícelo como display con animación y música.
ANIMACION	Los gráficos Sprite de Commodore le ayudan a crear sofisticados dibujos animados con 8 niveles distintos para moverlos por delante o detrás de los demás.
CUIDADO DE NINOS	El cartucho HOM BABYSITTER del COMMODORE 64 le ayudará a mantener a los niños ocupados durante horas aprendiendo a reconocer las letras del alfabeto. Además les enseñará los conceptos de parentesco.
PROGRAMACION EN BASIC	El MANUAL DE REFERENCIA DEL PROGRAMADOR y la serie de libros y cassetes APRENDA PROGRAMACION USTED MISMO le ofrece un excelente punto de partida.
NEGOCIOS	El COMMODORE 64 le ofrece una serie de programas de negocio de fácil uso, incluyendo el más potente procesador de textos y creador de impresos disponible en un ordenador personal.
COMUNICACION	Entre en el fascinante mundo de las redes de ordenadores. Conectando un VICMODEM a su COMMODORE 64 podrá comunicarse con otros usuarios de ordenadores en todo el mundo. Además podrá, en el futuro, suscribirse a los servicios de bancos de datos desde los que recibirá noticias, información financiera, servicios del hogar, etc. Usted podrá jugar con sus amigos conectando dos ordenadores entre sí.
COMPOSICION MUSICAL	El COMMODORE 64 está equipado con uno de los más sofisticados sintetizadores de música de los disponibles en los ordenadores personales. Tiene tres voces completamente programables, nueve octavas y cuatro tipos de onda controlables. Esté atento a los cartuchos y libros de música Commodore que le ayudarán a crear cualquier tipo de efectos de sonido.

CP/M*	Commodore ofrece opcionalmente el CP/M* en un cartucho conectable con el que podrá acceder a una extensa biblioteca de software.
DESARROLLO DE LA HABILIDAD	Coordinación Mano/Ojo y destreza manual pueden desarrollarse mediante muchos juegos Commodore, incluyendo "Aterrizaje en Júpiter", "Conducción de noche", etc.
EDUCACION	Aunque trabajar con un ordenador es en sí mismo una forma de educación, el libro de recursos educacionales COMMODORE contiene información general sobre el uso de ordenadores en educación. Existen además varios cartuchos diseñados para enseñar a todos desde música hasta matemáticas y de arte a astronomía.
IDIOMAS	El juego de caracteres programables del COMMODORE 64 le permite diseñar los caracteres de cualquier idioma extranjero.
GRAFICOS	Además de los gráficos Sprite mencionados antes, el COMMODORE 64 le ofrece alta resolución, gráficos multicolores por puntos, caracteres programables, y combinaciones de los diversos displays gráficos y de caracteres.
CONTROL DE INSTRUMENTOS	Su COMMODORE 64 tiene un port serie, un RS-232 y un port de usuario para su uso en una gran variedad de aplicaciones industriales. También está disponible opcionalmente un cartucho de interface IEEE/488.
CREACION DE CIRCULARES Y REVISTAS	El COMMODORE 64 le ofrece un excepcional procesador de textos que iguala o supera las prestaciones de los sistemas de más alto precio. Por supuesto usted puede guardar la información en la unidad de diskette 1541 o en el Datassette e imprimirla usando la impresora VIC PRINTER o un PLOTTER.
CONTROL DEL LAPIZ OPTICO	Las aplicaciones que requieran el uso de un lápiz óptico se pueden desarrollar utilizando cualquier lápiz óptico, y conectándolo en el port de juego de su COMMODORE 64.
PROGRAMACION EN LENGUAJE MAQUINA	El MANUAL DE REFERENCIA DEL PROGRAMADOR incluye una sección dedicada al lenguaje máquina, y otra que explica cómo utilizar rutinas en código máquina desde un programa en BASIC. Además hay una extensa bibliografía disponible para profundizar en su estudio.
NOMINAS, FACTURACION	El COMMODORE 64 puede ser programado con una gran variedad de aplicaciones de negocios. Los caracteres en mayúsculas y minúsculas, combinados con los "gráficos para negocios" C64 le permiten confeccionar fácilmente todo tipo de impresos, que puede imprimir en la VIC PRINTER u otra impresora.

*CPM es una marca registrada de Digital Research, Inc.

- Si precisa una introducción al uso del lenguaje máquina con programas BASIC nuestra revisión para principiantes le puede iniciar.
- Una potente característica de los ordenadores Commodore es el KERNAL. Esta ayuda le asegura que los programas que escriba hoy podrán ser usados por los ordenadores Commodore del mañana.
- La sección de programación de Entradas/Salidas le da la oportunidad de usar su ordenador hasta el límite. Se explica cómo conectar y usar desde lápices ópticos y joysticks hasta unidades de disco, impresoras y modems. (Periféricos de telecomunicaciones).
- Usted puede explorar el mundo de los SPRITES, caracteres programables y gráficos de alta resolución para obtener los más detallados y avanzados dibujos animados de la industria del microordenador.
- También puede entrar en el mundo de la sintetización musical y crear sus propias canciones y efectos de sonido con el mejor sintetizador de sonido incorporado a un ordenador personal.
- Si es usted un programador experto la sección de carga de soft le informa acerca de la posibilidad de utilizar en su COMMODORE 64 CP/M* y lenguajes de alto nivel, además del BASIC.

Piense que el MANUAL DE REFERENCIA DEL PROGRAMADOR es una útil herramienta que le ayudará en todo momento para que disfrute programando

COMO USAR EL MANUAL DE REFERENCIA DEL PROGRAMADOR

A lo largo del manual ha sido utilizada una notación convencional para describir la sintaxis (estructura de las sentencias de programación) de los comandos e instrucciones del BASIC, así como los requisitos y opciones de cada palabra reservada. Estas son las reglas para interpretar correctamente estas convenciones:

1. Las palabras reservadas del BASIC se muestran en mayúsculas. Las palabras reservadas deben escribirse tal como se muestran, sin añadir ni quitar nada.
2. La información entrecomillada indica datos variables que debe colocar usted. Ambas comillas y la información contenida en ellas deben escribirse tal como aparece en las explicaciones de cada instrucción.
3. El contenido entre llaves ([ ]) indica un parámetro opcional de la instrucción. Un parámetro es una limitación o calificación adicional de una instrucción. Si usa este parámetro opcional debe sustituir los datos por el parámetro opcional. Además, los puntos suspensivos le indican que puede repetir el parámetro tantas veces como lo permita la línea de programa.
4. Si un item entre llaves ([ ]) está escrito en MAYUSCULAS, le indica que DEBE utilizar estos determinados caracteres en el parámetro opcional, y deben escribirse tal como se muestran.
5. Los items entre llaves en ángulo (< >) indican datos variables que debe aportar. Mientras que la barra (/) indica que debe escoger entre dos opciones que se excluyen mutuamente.

*CP/M es una marca registrada de Digital Research Inc.

EJEMPLO DEL FORMATO DE SINTAXIS:

OPEN <núm.-fichero>, <periférico> [<dirección>], ["<unidad>: <nombre de fichero>] [, <modo>]"

EJEMPLOS DE LA INSTRUCCION:

```
10 OPEN 2,8,6,"0:STOCK FOLIO,S,W"
20 OPEN 1,1,2,"CHECKBOOK"
30 OPEN 3,4
```

Al aplicar estas convenciones a una situación práctica, las secuencias de los parámetros en sus instrucciones pueden no ser idénticas a las mostradas en los ejemplos de sintaxis. Los ejemplos no muestran todas las posibles secuencias. Se han intentado presentar todos los parámetros obligatorios y opcionales.
Los ejemplos de programación de este libro se presentan con espacios que separan las palabras y operadores para lograr una mayor legibilidad. Sin embargo, normalmente el BASIC no requiere espacios entre palabras a menos que su ausencia provoque ambigüedades o errores de sintaxis.
A continuación se muestran algunos ejemplos y descripciones de los símbolos usados para varios parámetros de instrucciones. La lista no contiene todas las posibilidades, pero le ayudará a comprender mejor cómo se presentan los ejemplos de sintaxis.

SIMBOLO	EJEMPLO	DESCRIPCION
<núm.-fichero>	50	Número de fichero lógico
<periférico>	4	Número de periférico físico
<dirección>	15	Número de dirección secundaria de periférico del bus serie
<unidad>	0	Número de la unidad de diskette
<nombre-fich.>	"TEST"	Nombre de fichero de datos o de programa
<constante>	"ABCDE"	Datos literales escritos por el programador
<variable>	X145	Nombre de cualquier variable o constante del BASIC
<cadena>	AB$	Se usa cuando se requiere una variable de cadena
<número>	12345	Se usa cuando se requiere una variable numérica
<num.-línea>	1000	Numero de línea del programa
<numérico>	1.5E4	Variable entera o de coma flotante

APLICACIONES DEL COMMODORE 64

Cuando pensó en comprar un ordenador probablemente se preguntaría, "Ahora puedo comprarme un ordenador, pero ¿qué puedo hacer con él?".
La mejor característica del COMMODORE 64 es que usted puede mandarle hacer ¡todo lo que desee! Usted puede calcular y archivar las cuentas de su casa o negocio. Puede usarlo como procesador de textos, para jugar a los más populares jue-

IMPRESION	Los interfaces del COMMODORE 64 le permiten la conexión de una gran variedad de impresoras de matriz y de alta calidad de impresión, así como plotters.
RECETAS DE COCINA	Usted puede almacenar en su COMMODORE 64 sus recetas favoritas, olvidándose de libros y tablas de medidas, ya que su ordenador le calculará rápidamente las cantidades necesarias según el número de comensales.
SIMULACIONES	Las simulaciones por ordenador le permiten la realización de experimentos peligrosos o caros con un mínimo riesgo y coste.
DATOS DEPORTIVOS	En un futuro próximo podrá conectar su ordenador a los bancos de datos centralizados, desde donde recibirá una completa información. Esto es posible gracias al COMMODORE 64 y al VICMODEM.
BOLSA DE VALORES	Conectando mediante un VICMODEM su ordenador a los futuros bancos de datos el COMMODORE 64 se convertirá en su agente de bolsa privado.

Estas son algunas de las aplicaciones de su COMMODORE 64. Como puede ver, en el trabajo o en el juego, en casa, en la escuela o en la oficina, su COMMODORE 64 le ofrece una solución práctica a todos sus problemas.
Commodore desea que conozca el soporte que ofrece a los usuarios de sus ordenadores. Se editan en los Estados Unidos dos publicaciones con informaciones de todo el mundo sobre los productos Commodore, existe ya un banco de datos al que conectar el ordenador para los usuarios de EE.UU. y Canadá.
Le recomendamos especialmente que se afilie al Club de Usuarios Commodore más cercano. Es una excelente forma de ampliar al máximo los conocimientos sobre su ordenador.
Por último, en la tienda donde compró su equipo podrán informarle de las novedades Commodore y resolverle cualquier posible problema.

POWER/PLAY
La Revista del Ordenador Personal

Revista de información sobre novedades, técnicas de programación, software, juegos, telecomunicaciones, etc. Publicada trimestralmente. Suscripción $10.00.

COMMODORE
La Revista del Microordenador

Pensada para educadores, científicos, empresarios... Contiene información sobre aplicaciones técnicas. Publicada bimensualmente. Suscripción $15.00.

RED DE INFORMACION COMMODORE

En España Microelectrónica y Control, distribuidor exclusivo de los productos Commodore, realiza un constante esfuerzo para proporcionar la mayor información a los usuarios de Commodore. Los manuales del usuario, así como los manuales de referencia e instrucciones de funcionamiento son traducidos rápidamente al castellano, además se presta toda la ayuda necesaria a los clubs de usuarios, fomentando su creación. Pero uno de los mejores logros es sin duda la revista mensual CLUB COMMODORE.

CLUB COMMODORE
Boletín Informativo para los Usuarios de Commodore

Esta revista aporta cada mes información interesante sobre técnicas de programación, trucos, nuevos productos y aplicaciones, programas, estructura interna de los ordenadores, etc. Además cuenta con espacio destinado a colaboraciones. Clubs de Usuarios, Bolsa de oportunidades, correo abierto... Es la conexión entre Commodore y los usuarios. Suscripción anual (11 números) 1.980 Ptas.

CAPITULO 1

REGLAS DE LA PROGRAMACION EN BASIC

- Introducción
- Códigos de pantalla
- Programación de números y variables
- Expresiones y operadores
- Técnicas de programación

INTRODUCCION

Este capítulo habla acerca de cómo el BASIC almacena y manipula los datos. Se incluyen los siguientes temas:

1) Una breve mención de los componentes y funciones del sistema operativo, como el juego de caracteres del Commodore 64.

2) La formación de constantes y variables. Tipos de variables. Cómo se almacenan en la memoria las constantes y variables.

3) Las reglas de los cálculos aritméticos, tests de comparación, tratamiento de cadenas y operaciones lógicas. Se incluyen también las reglas para formar expresiones, y las necesarias conversiones de datos cuando utiliza el BASIC con distintos tipos de datos mezclados.

CODIGOS DE PANTALLA (JUEGO DE CARACTERES DEL BASIC)

EL SISTEMA OPERATIVO (OS)

El sistema operativo está contenido en los chips de Memoria de Solo Lectura (ROM), y es una combinación de tres módulos de programas separados pero interrelacionados.

1) El Intérprete BASIC
2) El KERNAL
3) El Editor de Pantalla

1) **El Intérprete BASIC** es el responsable de analizar la sintaxis de las instrucciones y realizar los cálculos y manejo de datos. El intérprete BASIC tiene un vocabulario de 65 "palabras reservadas" con significados especiales. El alfabeto en mayúsculas y minúsculas así como los dígitos del cero al nueve se emplean para confeccionar las palabras reservadas y los nombres de variables. Algunos signos de puntuación y caracteres especiales tienen también significado para el intérprete. La tabla 1.1 muestra los caracteres especiales y sus usos.
2) **El KERNAL** trata muchos de los procesos de interrupción en el sistema (para detalles consulte el Capítulo 5). El KERNAL se encarga también de las Entradas/Salidas de datos.
3) **El Editor de Pantalla** controla la salida a la pantalla de video (o televisor) y edita el texto del programa BASIC. Además, el Editor de Pantalla reconoce las teclas que pulsa y decide si los caracteres que teclea se deben ejecutar en el acto o deben pasar al intérprete BASIC.

El Sistema Operativo le permite dos modos de operar en BASIC:
1) Modo DIRECTO
2) Modo PROGRAMA

1) Cuando está usando el modo DIRECTO, las instrucciones BASIC no tienen delante números de línea. Se ejecutan tan pronto se pulsa la tecla RETURN
2) El modo PROGRAMA es el que debe usar para ejecutar programas completos.

Tabla 1.1. Juego de caracteres CBM BASIC

CARACTER	NOMBRE Y DESCRIPCION
	ESPACIO. Separa palabras claves y nombres de variables
;	PUNTO Y COMA. Se usa en listas de variables para dar forma a las salidas
=	IGUAL. Asignador de valor y test de comparación
+	MAS. Adición aritmética y concatenación de cadenas. (concatenación enlazar varias cadenas en una)
-	MENOS. Substracción aritmética. Menos unario (-1)
*	ASTERISCO. Multiplicación aritmética. Menos unitario (-1)
/	BARRA. División aritmética
↑	FLECHA HACIA ARRIBA. Exponenciación aritmética
(	PARENTESIS IZQ. Evaluación de expresiones y funciones
)	PARENTESIS DER. Evaluación de expresiones y funciones
%	PORCENTAJE. Indica que la variable asignada es entera
#	NUMERO. Se coloca antes del número de fichero lógico en instrucciones de Entradas/Salidas
$	DOLAR. Indica que la variable asignada es una cadena
,	COMA. Se usa para formatear la salida de una lista de variables
.	PUNTO. Punto decimal en operaciones de coma flotante
"	COMILLAS. Delimitan las variables de cadena
:	DOS PUNTOS. Separa instrucciones BASIC en una línea
?	INTERROGANTE. Abreviación de la palabra clave PRINT
<	MENOR QUE. Usado en tests de comparación
>	MAYOR QUE. Usado en tests de comparación
π	PI. La constante numérica 3.141592654

Cuando se usa el modo PROGRAMA, todas las instrucciones BASIC deben tener un número de línea al principio de la misma. Usted puede colocar más de una instrucción BASIC en una línea de programa, pero el número de instrucciones queda limitado por la longitud total de la línea, que no puede exceder de 80 caracteres. Las instrucciones que no quepan en una línea deben colocarse en una nueva línea que se iniciará con un nuevo número de línea.
El Commodore 64 posee dos juegos de caracteres completos, que puede usar directamente desde el teclado o dentro de sus programas.
En el primer juego el alfabeto en mayúsculas y los números del 0 al 9 están disponibles sin necesidad de pulsar la tecla SHIFT. Si aprieta la tecla SHIFT al pulsar un carácter, se mostrará el carácter gráfico situado a la derecha de la tecla. Si usa la tecla C=* al pulsar una tecla se mostrará el carácter gráfico de la izquierda de la tecla pulsada. Al pulsar SHIFT y una tecla que no contenga un símbolo gráfico aparecerá el signo colocado en la parte de arriba de la cara superior de la tecla
En el segundo juego el alfabeto en minúsculas y los dígitos del 0 al 9 se podrán escribir sin necesidad de pulsar la tecla SHIFT. El alfabeto en mayúsculas está disponible pulsando la tecla SHIFT simultáneamente con la tecla correspondiente a la tecla escogida. También se pueden mostrar los símbolos gráficos de la izquierda de las teclas pulsando la tecla C= junto con la del carácter escogido. Los símbolos de

* Este signo indica la tecla con el logotipo Commodore que está en la parte inferior izquierda del teclado

la parte de arriba de la cara superior de las teclas se obtienen pulsando SHIFT junto con la tecla deseada.
Para cambiar de un juego de caracteres al otro pulse simultáneamente las teclas SHIFT y C=

PROGRAMACION DE NUMEROS Y VARIABLES

CONSTANTES ENTERAS, DE COMA FLOTANTE Y DE CADENA

Las constantes son datos que usted coloca en sus instrucciones BASIC. El BASIC utiliza los valores que representan los datos durante la ejecución de la instrucción. El CBM BASIC reconoce tres tipos de constantes:

1) NUMEROS ENTEROS
2) NUMEROS EN COMA FLOTANTE
3) CADENAS

Las constantes enteras son números enteros (números sin punto decimal). Estas constantes deben estar comprendidas entre los números −32768 y +32768. Las constantes enteras no deben tener puntos decimales o comas entre los dígitos. Si se omite el signo + a la izquierda del número se asume que la constante es positiva. Los ceros a la izquierda de la constante son ignorados, ya que gastarían memoria y enlentecerían el programa. Sin embargo, esto no es causa de error. Los enteros se almacenan en memoria en forma de número binario de dos bytes. He aquí algunos ejemplos de constantes enteras:

```
   -12
  8765
-32768
   +44
     0
-32767
```

> NOTA: No coloque comas entre los dígitos de un número. Por ejemplo, el número 32,000 debe escribirse 32000. Si coloca una coma entre dos dígitos recibirá el mensaje de error BASIC: ?SINTAX ERROR.

Constantes de coma flotante son números positivos o negativos que pueden contener decimales. Recuerde que debe utilizar el punto y no la coma para fraccionar el número. Si el signo + no está presente a la izquierda de un número el Commodore 64 asume que este es positivo. Si omite el punto decimal el ordenador asume que se encuentra detrás del último dígito. Como en el caso de las constantes enteras, los ceros a la izquierda son ignorados. Las constantes de coma flotante se pueden usar de dos modos:

1) NUMERO SIMPLE
2) NOTACION CIENTIFICA

Las constantes enteras se representan en su televisor con nueve cifras significativas, es decir, valores de −999999999 a +999999999. Si entra un número mayor, éste se redondea basándose en el último dígito: si éste es mayor o igual a 5 se redondea hacia arriba. Si el número es menor de 5 se redondea hacia abajo. Los números en coma flotante se almacenan (usando cinco bytes de memoria) y se manipulan con diez dígitos de precisión. Sin embargo, los números se redondean a nueve dígitos cuando se imprimen los resultados.

He aquí algunos ejemplos de números en coma flotante:

```
  1.23
  -.998877
 +3.1459
   .7777777
-333.
   .01
```

Los números más pequeños de .01 o más grandes de 999999999 se imprimen utilizando la notación científica. En la notación científica un número en coma flotante se compone de tres partes:

1) LA MANTISA
2) LA LETRA E
3) EL EXPONENTE

La mantisa es un número en coma flotante. La letra E se utiliza para decirle que está viendo un número en forma exponencial. En otras palabras, E representa *10 (ej. 3E3=3*10 ↑ 3=3000). Y el exponente indica cuántas veces se debe multiplicar el 10 para obtener el valor del número.
Tanto la mantisa como el exponente pueden ser números positivos o negativos. El exponente puede tener un valor desde −39 hasta +38 y ello indica los lugares que el punto decimal de la mantisa debe correr hacia la izquierda (−) o hacia la derecha (+).
Hay un límite en el tamaño de los números, incluso en notación científica, que el BASIC puede manipular: el número más alto es +1.70141183E+38, y los cálculos que arrojen un resultado mayor generarán el mensaje de error ?OVERFLOW ERROR. El número más pequeño es +2.93873588E−39, y las operaciones en las que se obtenga un número menor darán como resultado cero, sin generar un mensaje de error. A continuación se muestran algunos ejemplos de números en notación científica (y su valor decimal):

253.988E−3	(.253988)
2359E6	(2359000000.)
−7.09E−12	(−.00000000000709)
−3.14159E+5	(−314159.)

Las constantes de cadena son grupos de información alfanumérica, como letras, números y símbolos. Cuando entra una cadena desde el teclado ésta puede tener cualquier longitud hasta el espacio disponible de 80 caracteres que tiene una línea. Una constante de cadena puede contener espacios, letras, números, puntuación y caracteres de control de color o del cursor. Usted puede colocar comas entre los números. El único carácter que no puede colocar son las comillas. Esto es debido a que las comillas sirven para definir el principio y fin de una cadena. Una cadena puede también estar vacía, es decir, no contener ningún dato. Usted puede no

incluir las comillas de fin de cadena si ésta es la última parte de una línea o si está seguida de dos puntos (:). He aquí algunos ejemplos de constantes de cadena:

"" (cadena vacía)
"HOLA"
"$25.000,00"
"NUMERO DE EMPLEADOS"

Nota: Use CHR$(34) para incluir comillas en las cadenas.

VARIABLES ENTERAS, DE COMA FLOTANTE Y DE CADENA

Las variables son nombres que representan datos usados en sus instrucciones BASIC. El valor representado por una variable puede asignarse colocando un igual en una constante, o puede ser el resultado de cálculos del programa. Las variables, al igual que las constantes, pueden ser enteras, de coma flotante o cadenas. Si usted se refiere a un nombre de variable en el programa antes de que le haya asignado un valor, el intérprete BASIC crea automáticamente la variable con un valor de cero si es una variable entera o de coma flotante. O bien crea una cadena vacía si emplea una variable de cadena.
Los nombres de las variables pueden tener cualquier longitud, pero sólo los dos primeros caracteres son significativos en el CBM BASIC. Esto significa que los nombres utilizados para variables no deben tener los dos primeros caracteres iguales. Los nombres de variables no pueden ser iguales a alguna palabra reservada del BASIC, y tampoco pueden contener palabras reservadas en medio del nombre. Las palabras reservadas del BASIC incluyen comandos, instrucciones, nombres de función y operadores lógicos. Si usa accidentalmente una palabra reservada en medio del nombre de una variable aparecerá el mensaje de error BASIC ?SYNTAX ERROR.

Los caracteres utilizados para formar nombres de variables son el alfabeto y los números del cero al nueve. El primer carácter de un nombre debe ser una letra. Los caracteres de declaración del tipo de variable (%) y ($) pueden ser usados como último carácter del nombre. El signo (%) indica que la variable es entera y el signo ($) indica una variable de cadena. Si no se incluye un carácter de declaración el intérprete asume que la variable es de coma flotante. A continuación se indican algunos ejemplos de nombres de variables, sus valores, y el tipo de datos:

A$="GRAN VENTA" (variable de cadena)
MES$="ENE"+A$ (variable de cadena)
K%=5 (variable entera)
CNT%=CNT%+1 (variable entera)
FP=12.5 (variable de coma flotante)
SUM=FP*CNT% (variable de coma flotante)

TABLAS ENTERAS, DE COMA FLOTANTE Y DE CADENA

Un array es una tabla (o lista) de datos asociados referidos a un solo nombre de variable. En otras palabras, una tabla es una secuencia de variables, por ejemplo una lista de números. Cada número individual de la tabla es un elemento del array. Las tablas son útiles para describir un largo número de variables. Imagínese que la tabla tiene diez filas de números con veinte números en cada fila. Esto hace un total de doscientos numeros en la tabla. Sin utilizar un array debería utilizar doscientos nombres de variable. Pero gracias al uso de arrays, usted sólo necesita un nombre para la tabla y todos los elementos contenidos en ella se identifican por su posición en la misma.
Los nombres de tablas pueden ser enteros, de coma flotante o de cadena y todos los elementos de la tabla deben ser del tipo de datos especificado en el nombre de la tabla. Las tablas pueden tener una sola dimensión (una simple lista) o múltiples dimensiones (imagine una malla marcada en filas y columnas, o un cubo de Rubik).
Cada elemento de una tabla se identifica por un índice de variable que sigue al nombre de array, encerrado en paréntesis.
El máximo número de dimensiones de una tabla es en teoría doscientos cincuenta y cinco y el número de elementos en cada dimensión se limita a 32.767. Pero para los propósitos prácticos, el tamaño de las tablas queda limitado por el espacio disponible en memoria y/o la línea lógica de 80 caracteres. Si un array tiene sólo una dimensión y si el indice de variable no excede de 10 (tabla de 11 elementos: de 0 a 10) entonces el array es creado por el intérprete, asignando a cada elemento el valor cero (o cadena vacía si es array de cadena) la primera vez que el programa se refiera a él. Si el array sobrepasa los 11 elementos debe usarse la instrucción BASIC DIM, para definir el nombre, tipo y tamaño de la tabla. La memoria necesaria para contener una tabla se puede calcular teniendo en cuenta lo siguiente:

	5 bytes del nombre de la tabla
+	2 bytes por cada dimensión de la tabla
+	2 bytes por cada elemento en tablas enteras
o +	5 bytes por cada elemento en coma flotante
o +	3 bytes por cada elemento en cadenas
y +	1 byte por carácter en cada elemento de cadena

Los índices pueden ser constantes enteras, variables, o una expresión aritmética que de como resultado un número entero. Es necesario un índice por cada dimensión de la tabla. Los diversos índices se separaran por comas. Los índices deben tener un valor comprendido entre cero y el número de elementos de la tabla. Los valores fuera de este rango generarán el mensaje de error BASIC ?BAD SUBSCRIPT.
He aquí algunos ejemplos de nombres de tablas y sus tipos de datos.

A$(0)="GRAN VENTA"	(tabla de cadenas)
MES$(K%)="ENE"	(tabla de cadenas)
G2%(X)=5	(tabla de enteros)
CNT%(G2%(X))=CNT%(1)-2	(tabla de enteros)
FP(12*K%)=24.8	(tabla de coma flotante)
SUM(CNT%(1))=FP*K%	(tabla de coma flotante)

A(5)=0 (Asigna el valor cero al quinto elemento de la tabla unidimensional llamada A)

B(5,6)=0 (Asigna el valor cero al elemento colocado en la fila 5 y columna 6 de la tabla de dos dimensiones llamada B)

C(1,2,3)=0 (Asigna el valor cero al elemento situado en la fila 1, columna 2 y profundidad 3 en la tabla denominada C)

EXPRESIONES Y OPERADORES

Las expresiones se forman usando constantes, variables y/o tablas. Una expresión puede ser una sola constante, variable o una variable de tabla de cualquier tipo. También puede ser una combinación de constantes y variables con operadores aritméticos, lógicos o de comparación diseñados para producir un solo valor. Más adelante se explica como trabajan los operadores. Las expresiones se pueden dividir en dos clases:

1) ARITMETICAS
2) DE CADENA

Las expresiones tienen normalmente dos o más datos llamados operandos. Cada operando está separado por un solo operador para producir el resultado deseado. Normalmente se asigna el valor de la expresion a un nombre de variable. Todos los ejemplos de constantes y variables que ha visto ya son también ejemplos de expresiones.

Un operador es un símbolo especial reconocido por el intérprete BASIC del Commodore 64, y representa una operación a realizar entre variables o constantes. Uno o más operadores, combinados con una o más constantes y/o variables, forman una expresión. El BASIC del Commodore 64 reconoce operadores aritméticos, lógicos y de comparación.

EXPRESIONES ARITMETICAS

El resultado de las expresiones aritméticas da un número entero o de coma flotante. Los operadores aritméticos (+,/,−, *, ↑) se usan para realizar suma, división, resta, multiplicación y exponenciación respectivamente.

OPERACIONES ARITMETICAS

Un operador aritmético define una operación aritmética realizada entre dos operandos, uno a cada lado del operador. Las operaciones aritméticas se realizan usando números en coma flotante. Los enteros se convierten en números de coma flotante antes de realizar una operación aritmética. El resultado se convierte de nuevo en entero si éste es asignado a un nombre de variable de este tipo.

SUMA (+): El signo más (+) especifica que el operando de la derecha se añade al operando de la izquierda.

EJEMPLOS:

2+2
A+B+C
X%+1
BR+10E−2

RESTA (−): El signo menos especifica la sustracción del operando de la derecha al operando de la izquierda.

EJEMPLOS:

4−1
100−64
A−B
55−142

El signo menos puede ser usado para números negativos. Se entiende de esta forma cuando el signo está delante de un número. Esto es igual a sustraer dicho número de cero (0).

EJEMPLOS:

−5
−9E4
−B
4−(−2) igual a 4+2

MULTIPLICACION (*): El asterisco (*) especifica que el operando de la izquierda es multiplicado por el operando de la derecha.

EJEMPLOS:

100*2
50*0
A*X1
R%*14

DIVISION (/): La barra (/) indica que el operando de la izquierda es dividido por el operando de la derecha.

EJEMPLOS:

10/2
6400/4
A/B
4E2/XR

EXPONENCIACION (↑): La flecha hacia arriba (↑) especifica que el operando de la izquierda se eleva a la potencia indicada por el operando de la derecha (o exponente). Si el operando de la derecha es 2, el número de la izquierda se eleva al cuadrado. Si el exponente es 3, el número se eleva al cubo, etc. El exponente puede ser cualquier número, en tanto el resultado sea un número en coma flotante válido.

EJEMPLOS:

2 ↑ 2 Equivale a 2*2
3 ↑ 3 Equivale a 3*3*3
4 ↑ 4 Equivale a 4*4*4*4
AB ↑ CD
3 ↑ -2 Equivale a 1/3*1/3

OPERADORES DE COMPARACION

Los operadores de comparación (<,=,>,<=,>=,<>) se usan principalmente para comparar los valores de dos operandos, pero también producen un resultado aritmético. Los operadores de comparación y los operadores lógicos (AND,OR, y NOT), cuando se usan en comparaciones producen un valor aritmético verdadero/falso al evaluar la expresión. Si la relación entre los operandos es cierta se produce el valor entero –1, y si es falsa se produce el valor 0. Estos son los operadores de comparación:

<	MENOR QUE
=	IGUAL A
>	MAYOR QUE
<=	MENOR O IGUAL QUE
>=	MAYOR O IGUAL QUE
<>	NO IGUAL A

EJEMPLOS:

1=5-4 verdadero (–1)
14>66 falso (0)
15>=15 verdadero (–1)

Los operadores de comparación pueden usarse para comparar cadenas. Para los propósitos de comparación, las letras del alfabeto tienen el orden A<B<C<D, etc. Las cadenas se comparan evaluando el orden entre los caracteres correspondientes de izquierda a derecha (véase Operaciones con Cadenas).

EJEMPLOS:

"A"<"B" verdadero (–1)
"X"="YY" falso (0)
BB$<>CC$

Los datos numéricos sólo se pueden comparar con otros datos numéricos. Lo mismo ocurre en la comparación de cadenas, si no se sigue esta regla aparecerá el mensaje de error BASIC **?TYPE MISMATCH**. En la comparación de operandos el ordenador convierte uno o ambos operandos en números de coma flotante si ello es necesario. Entonces se comparan los valores en coma flotante para obtener el resultado verdadero/falso.
Al fin de las comparaciones se obtiene un entero independientemente del tipo de datos comparados, incluso si son cadenas. A causa de esto, la comparación de valores puede ser utilizada como operando de un cálculo. El resultado –1 o 0 puede ser usado en cualquier operación, menos como divisor, ya que la división por cero es ilegal.

OPERADORES LOGICOS

Los operadores lógicos (AND, OR, NOT) pueden ser usados para modificar el resultado de los operadores de comparación o para producir un resultado aritmético. Los operadores lógicos pueden producir otros números además de –1 y 0, pero cualquier resultado que no sea cero se considera como verdadero en el test verdadero/falso.
Los operadores lógicos (llamados a veces operadores Booleanos) pueden también ser usados para realizar operaciones sobre un dígito binario individual (bit) en dos operandos. Pero cuando use el operador NOT, la operación se realiza sólo en el operando de la derecha. Los operandos deben ser números enteros (de –32768 a +32767) (los números en coma flotante se convierten en enteros) y las operaciones lógicas dan un resultado entero.
Los operadores lógicos trabajan comparando bit por bit en los dos operandos. El operador AND produce un resultado de 1 sólo si los correspondientes bits de los operandos tienen ambos un valor de 1. El operador lógico OR produce un resultado de 1 si cualquiera de los dos operandos tienen el bit a 1. El operador lógico NOT es el valor opuesto a cada bit del operando. En otras palabras, NOT 1 equivale a 0 y NOT 0 equivale a 1.
El OR exclusivo (XOR) no es un operador lógico, pero forma parte de la instrucción WAIT. El OR exclusivo significa que si los bits de los dos operandos son iguales el resultado es 0 y si no el resultado es 1.
Las operaciones lógicas se han definido como grupos de instrucciones que constituyen la tabla Booleana mostrada en la Tabla 1.2

Los operadores lógicos AND, OR y NOT especifican una operación BOOLEANA a efectuar entre los operandos a cada lado del operador. En el caso de NOT, SOLO se considera el operando de la derecha. Las operaciones lógicas (o aritmética Booleana) no se realizan hasta haber completado todas las operaciones aritméticas y de comparación.

EJEMPLOS:

IF A=100 AND B=100 THEN 10	(si A y B tienen el valor de cien el resultado es cierto)
A=96 AND 32: PRINT A	(A=32)

Tabla 1.2. Tabla Booleana.

La operación AND da 1 sólo si ambos bits son 1:

```
1 AND 1 = 1
0 AND 1 = 0
1 AND 0 = 0
0 AND 0 = 0
```

La operación OR da 1 si algún bit es 1:

```
1 OR 1 = 1
0 OR 1 = 1
1 OR 0 = 1
0 OR 0 = 0
```

La operación NOT complementa cada bit:

```
NOT 1 = 0
NOT 0 = 1
```

El OR exclusivo (XOR) es parte de la instruccion WAIT:

```
1 XOR 1 = 0
1 XOR 0 = 1
0 XOR 1 = 1
0 XOR 0 = 0
```

IF A=100 OR B=100 THEN 20	(Si A o B valen cien, el resultado es verdadero)
A=64 OR 32: PRINT A	(A=96)
IF NOT X<Y THEN 30	(si X>Y el resultado es verdadero)
X=NOT 96	(el resultado es –97 (complementario))

JERARQUIA DE LAS OPERACIONES

Todas las expresiones realizan las distintas operaciones de acuerdo con una jerarquía fija. En otras palabras, algunas operaciones se realizan antes que otras. El orden normal de las operaciones se puede modificar colocando dos o más operandos entre paréntesis (), creando una "subexpresión". Las partes encerradas entre paréntesis se reducen a un solo valor antes de trabajar con las partes fuera del paréntesis.

Cuando use paréntesis en sus expresiones debe siempre colocar el mismo número de ellos para abrir y para cerrar, sino aparecerá el mensaje de error BASIC **?SYNTAX ERROR.**

Las expresiones que contengan operandos entre paréntesis pueden estar ellas mismas entre paréntesis, formando expresiones complejas de múltiples niveles. Se pueden crear expresiones con hasta 10 niveles. (Diez pares de paréntesis). Las operaciones con los paréntesis más interiores se ejecutarán primero. Vea algunos ejemplos de expresiones:

```
A+B
C↑(D+E)/2
((X–C↑(D+E)/2)*10)+1
GG$>HH$
JJ$+"MAS"
K%=1 AND M<>X
K%=2 OR (A=B AND M<X)
NOT (D=E)
```

El intérprete BASIC normalmente realiza las operaciones de cada expresión ejecutando primero las aritméticas, después las de comparación y por último las operaciones lógicas. Los operadores aritméticos y lógicos tienen un orden de ejecución (o jerarquía de las operaciones) entre ellas mismas. Por otra parte, los operadores de comparación no tienen este orden, y se ejecutan de izquierda a derecha. Todos los operadores de una expresión que tengan la misma jerarquía se ejecutarán de izquierda a derecha. Al realizar operaciones entre paréntesis, dentro del mismo se mantiene el orden normal de ejecución. La jerarquía de las operaciones se muestra en la Tabla 1.3 de la más alta a la más baja.

Tabla 1.3. Jerarquía de las operaciones en una expresion

OPERADOR	DESCRIPCION	EJEMPLO
↑	Exponenciación	BASE ↑ EXP
–	Negación (menos unario)	–A
*/	Multiplicación/División	AB*CD EF/GH
+-	Suma/Resta	CN+2 JK-PQ
>=<	Operadores de comparación	A <= B
NOT	NOT lógico (complementa enteros)	NOT K%
AND	AND lógico	KJ AND 128
OR	OR lógico	PQ or 15

OPERACIONES DE CADENAS

Las cadenas se comparan usando los mismos operadores de comparación (=,<>, <=,>=,<,>) que con los números. Las comparaciones de cadenas se realizan tomando un carácter a la vez (de izquierda a derecha) de cada cadena y evaluando el código de carácter del juego de caracteres PET/CBM. Si el código es igual, el carácter es igual. Si un código difiere del otro, el carácter con el código menor es el menor en el juego de caracteres. La comparación finaliza al terminar la cadena. A igualdad de caracteres, se considera menor la cadena más corta. Los espacios y caracteres de control SON significativos.

Al igual que con los otros tipos de datos, al finalizar la comparación se produce un resultado entero. Esto ocurre si los dos operandos son cadenas. Usted puede utilizar el resultado para realizar operaciones. El resultado será -1 o 0 (verdadero/falso). El resultado no puede ser utilizado como divisor, ya que la división por cero es ilegal.

EXPRESIONES DE CADENA

Las expresiones son tratadas como si un "<>0" implícito las siguiera. Esto significa que si una expresión es verdadera se ejecuta la próxima instrucción BASIC del programa. Si la expresión es falsa se ignora el resto de la línea y se pasa a la siguiente. Además de con números, usted puede realizar operaciones con variables de cadena. El único operador aritmético de cadenas reconocido por el CBM BASIC es el signo más (+), usado para concatenar cadenas. Al concatenar dos cadenas, la de la derecha se añade a continuacion de la de la izquierda, formando como resultado una tercera cadena. El resultado se puede imprimir, utilizar en comparaciones, o ser asignado a un nombre de variable. Si una cadena se compara con un número o viceversa, aparece el mensaje de error BASIC **?TYPE MISMATCH**. Algunos ejemplos de expresiones de cadena y concatenaciones se muestran a continuación:

```
10 A$="FICHERO":B$="NOMBRE"
20 NOM$=B$+A$              (da la cadena: NOMBREFICHERO
30 RES$="NUEVO "+B$+A$     (da la cadena: NUEVO NOMBRE
                           FICHERO)
```

Fíjese en este espacio.

TECNICAS DE PROGRAMACION

CONVERSION DE DATOS

Cuando es necesario, el intérprete CBM BASIC puede convertir un dato numérico entero a coma flotante, y viceversa, de acuerdo con las siguientes reglas:

- Todas las operaciones aritméticas y de comparacion se ejecutan en formato de coma flotante. Los enteros se convierten en números de coma flotante para evaluar la expresión, y el resultado se convierte de nuevo en entero. Las operaciones lógicas convierten los operandos en enteros y dan el resultado entero.
- Si un dato numérico de un tipo se almacena en un nombre de variable de distinto tipo se convierte el dato al tipo especificado en el nombre de la variable.
- Cuando un número en coma flotante se convierte en entero se trunca la parte decimal (se elimina), y el entero queda menor o igual que el número en coma flotante. Si el resultado está fuera del rango -32768 a +32767 aparecerá el mensaje de error BASIC **?ILLEGAL QUANTITY.**

USO DE LA INSTRUCCION INPUT

Ahora que conoce los diversos tipos de variables, lea esta información y combínela con lo visto anteriormente para conseguir programas prácticos.
En nuestro primer ejemplo usted debe pensar que una variable es una "celda de almacenamiento" donde el Commodore 64 almacena la respuesta del usuario a la pregunta formulada. Para escribir un programa que pida al usuario que escriba su nombre, debe asignar la variable N$ al nombre escrito. Ahora, cada vez que el programa tenga la instrucción PRINT N$, el Commodore 64 IMPRIMIRA automáticamente el nombre que haya escrito el usuario.
Escriba la palabra NEW en su Commodore 64. Pulse la tecla RETURN, y escriba este ejemplo:

```
10 PRINT"SU NOMBRE":INPUT N$
20 PRINT"HOLA,"N$
```

En este ejemplo usted utiliza N para recordar que esta variable se refiere al "NOMBRE". El signo del dólar ($) se usa para indicar al ordenador que es una variable de cadena. Es importante distinguir entre los dos tipos de variables:

1) NUMERICAS
2) DE CADENA

Usted probablemente recuerda de las anteriores secciones que las variables numéricas se usan para almacenar números como 1, 100, 4000, etc. Una variable numérica puede ser una sola letra (A), cualquier combinación de dos letras (AB), una letra y un número (A1), o dos letras y un número (AB1). Ahorrará memoria usando nombres de variables cortos. Otro punto interesante es utilizar letras y números para las diversas categorias en un mismo programa (A1,A2,A3). También, si necesita números enteros en lugar de números con punto decimal, debe colocar el signo de tanto por ciento (%) al final del nombre de variable (AB%,A1%,etc.)
Vea ahora unos ejemplos que utilizan distintos tipos de variables y expresiones con la instrucción INPUT.

```
10 PRINT"ENTRE UN NUMERO":INPUT A
20 PRINT A

10 PRINT"ENTRE UNA PALABRA":INPUT A$
20 PRINT A$

10 PRINT"ENTRE UN NUMERO":INPUT A
20 PRINT A "POR CINCO IGUAL A" A*5
```

NOTA: El ejemplo 3 le muestra que los mensajes a imprimir van entre comillas, mientras que las variables no. Tome nota también de que en la línea 20 se imprime primero la variable A, después el mensaje "POR CINCO IGUAL A", y entonces el resultado de multiplicar la variable A por 5 (A*5).

Los cálculos son importantes en la mayoría de los programas. Usted puede escoger entre usar "números efectivos" o variables para los cálculos, pero si trabaja con números proporcionados por el usuario debe usar variables numéricas. Empiece por pedir al usuario que escriba dos números de esta forma:

```
10 PRINT"ESCRIBA 2 Números":INPUT A:INPUT B
```

EJEMPLO DE PRESUPUESTO DE INGRESOS/GASTOS

```
5 PRINT "♥"  ← SHIFT CLR/HOME
10 PRINT "INGRESOS MENSUALES":INPUT IN
20 PRINT
30 PRINT "GASTOS CATEGORIA 1":INPUT E1$
40 PRINT "TOTAL GASTOS":INPUT E1
50 PRINT
60 PRINT "GASTOS CATEGORIA 2":INPUT E2$
70 PRINT "TOTAL GASTOS":INPUT E2
80 PRINT
90 PRINT [illegible] CATEGORIA 3":INPUT E3$
100 PRINT [illegible] GASTOS":INPUT E3
110 PRINT [illegible] SHIFT CLR/HOME
120 E=E1+[illegible]
130 EP=E/IN
140 PRINT "INGRESOS MENSUALES: $"IN
150 PRINT "GASTOS TOTALES:  $"E
160 PRINT"BALANCE:  $"IN-E
170 PRINT
180 PRINTE1$"="(E1/E)*100"% DEL TOTAL DE GASTOS"
190 PRINTE2$"="(E2/E)*100"% DEL TOTAL DE GASTOS"
200 PRINTE3$"="(E3/E)*100"% DEL TOTAL DE GASTOS"
210 PRINT
220 PRINT"SUS GASTOS="EP*100"% DE SUS INGRESOS TOTALES"
230 FOR X=1 TO 5000:NEXT:PRINT
240 PRINT"OTRA VEZ? (S O N)":INPUT Y$:IF Y$="S" THEN 5
250 PRINT"♥":END  ← SHIFT CLR/HOME
READY.
```

NOTA: IN no PUEDE ser igual a cero, y E1, E2, E3 no PUEDEN ser cero los tres a la vez.

DESCRIPCION LINEA A LINEA DEL EJEMPLO DE PRESUPUESTO DE INGRESOS/GASTOS

Linea(S)	Descripción
5	Limpia la pantalla.
10	Instrucciones PRINT/INPUT.
20	Inserta una línea en blanco.
30	Gastos. Categoría 1=E1$.
40	Cantidad gastada=E1.
50	Inserta una línea en blanco.
60	Gastos. Categoría 2=E2$.
70	Cantidad gastada=E2.
80	Inserta una línea en blanco.
90	Gastos. Categoría 3=E3$.
100	Cantidad gastada=E3.
110	Limpia la pantalla.
120	Suma todos los gastos=E.
130	Calcula Gastos/Ingresos%.
140	Display de ingresos.
150	Display de gastos totales.
160	Display de ingresos-gastos.
170	Inserta línea en blanco.
180-200	Estas líneas calculan el % de cada cuenta de gastos sobre el total de gastos.
210	Inserta una línea en blanco.
220	Display G/I%.
230	Pausa temporal.

Ahora multiplique estos números para crear una nueva variable llamada C, tal como se muestra en la línea 20:

```
20 C=A*B
```

Para imprimir el resultado junto con un mensaje escriba:

```
30 PRINT A "POR" B "IGUAL A" C
```

Entre estas 3 líneas y ejecute el programa mediante el comando RUN. Recuerde que los mensajes van entre comillas, y las variables no.

Ahora recuerde que necesita colocar el signo del dólar ($) delante del número representado por la variable C. El signo $ debe ser impreso dentro de comillas y antes de la variable C. Para añadir este símbolo a su programa pulse la tecla **RUN/STOP** y **RESTORE** simultáneamente. Ahora escriba la línea 40 como sigue:

```
40 PRINT"$"C
```

Ahora pulse **RETURN**, escriba RUN y pulse **RETURN** otra vez.
El signo $ debe ir entre comillas porque la variable C solo representa un número y no puede contener $. Si el número contenido en C es 100 el Commodore 64 le mostrará en la pantalla $ 100. Pero, si usted intenta PRINT $C sin usar comillas, recibirá el mensaje de error **?SYNTAX ERROR**.

Un último comentario acerca del signo $. Usted puede crear una variable que represente el signo del dólar que podrá sustituir al $ cuando lo necesite en variables numéricas. Por ejemplo:

```
10 Z$="$"
```

Ahora cada vez que necesite el signo del dólar puede utilizar la variable de cadena z$. Pruebe esto:

```
10 Z$="$":INPUT A
20 PRINT Z$A
```

La línea 10 define el signo $ como una variable llamada Z$, y luego pide un numero llamado A. La línea 20 imprime Z$ ($) y después A (numero).
Usted probablemente encontrará más sencillo asignar signos como el dólar —o mejor aun Ptas.— a una variable, puesto que cada vez que tenga que utilizarlos bastará con escribir solo el nombre de la variable en lugar de los símbolos y las comillas.

USO DE LA INSTRUCCION GET

Muchos programas utilizan INPUT para que el usuario entre datos en el ordenador. Si tiene necesidades más complejas, como comprobar que no se entren errores de escritura, la instrucción GET da más flexibilidad e "inteligencia" a su programa. Esta seccion le muestra como usar la instruccion GET para añadir caracteristicas especiales de edición de pantalla a sus programas.
El Commodore 64 tiene un buffer de teclado en el que caben hasta diez caracteres. Esto significa que si el ordenador está ocupado en alguna operación que no sea leer el teclado, usted puede escribir hasta diez caracteres, que el Commodore 64 usará tan pronto como termine su trabajo. Para demostrar esto, escriba este programa en su Commodore 64:

```
10 TI$="000000"
20 IF TI$<"000015"THEN20
```

Ahora escriba RUN, pulse RETURN y mientras el programa se ejecuta, escriba la palabra HOLA.
Advierta que no sucede nada durante quince segundos desde que se puso en marcha el programa. Después de este lapso aparecerá el mensaje HOLA en la pantalla.
Imagínese haciendo cola para ver una pelicula. La primera persona de la cola es la primera en coger su entrada y dejar la cola. La última persona es también la última en coger su entrada. La instrucción GET funciona de una manera parecida a la taquillera. Primero mira si hay algún carácter "en la cola". En otras palabras, mira si se ha pulsado alguna tecla. Si es así, el carácter pulsado se coloca en la variable apropiada. Si no se ha pulsado ninguna tecla, se le asigna un valor vacio a la variable.
En este punto es importante que advierta que si intenta escribir más de diez caracteres a la vez en el buffer, todos los caracteres a partir del décimo se pierden.

Puesto que la instrucción GET funciona aunque no se pulse ningún carácter, es necesario colocarla en un bucle para que espere hasta que reciba un carácter.
A continuación se muestra la forma recomendada para usar la instrucción GET. Escriba NEW para borrar los programas previos.

```
10 GET A$ : IF A$ = "" THEN 10
```

Advierta que no debe colocar ningún espacio entre las comillas. Esto indica un valor vacio y obliga al programa a seguir ejecutando la instrucción GET hasta que se pulsa una tecla. Solo cuando una tecla es pulsada, el programa continua con la linea siguiente a la diez. Añada esta línea a su programa:

```
100 PRINT A$;: GOTO 10
```

Ahora haga un RUN. Advierta que el cursor no aparece en la pantalla, pero cada carácter que escriba se muestra en ella. La segunda línea del programa puede convertirse en una parte de un editor de pantalla, como se demuestra más adelante. Hay muchas cosas que usted puede hacer con un editor de pantalla. Puede tener un cursor parpadeante, puede proteger teclas como CLR/HOME para que no se borre accidentalmente la pantalla. Puede también hacer que las teclas de función impriman palabras o frases. El siguiente programa asigna a cada tecla de función un determinado propósito. Recuerde que esto es sólo el principio de un programa que usted puede adaptar a sus necesidades.

```
20 IF A$ = CHR$(133) THEN POKE 53280,8:GOTO10
30 IF A$ = CHR$(134) THEN POKE 53281,4:GOTO10
40 IF A$ = CHR$(135) THEN A$="MUY SEÑOR MIO:"+CHR$(13)
50 IF A$ = CHR$(136) THEN A$="ATENTAMENTE,"+CHR$(13)
```

Los números entre paréntesis en CHR$ corresponden a los de la tabla CHR$ del apéndice C. La tabla muestra un número distinto para cada carácter. Las cuatro teclas de función se han inicializado para realizar el trabajo descrito por las instrucciones que siguen a la palabra THEN en cada línea. Cambiando el número de CHR$ puede usted utilizar distintas teclas. Pueden utilizarse para distintos trabajos cambiando la información contenida después de las instrucciones THEN.

COMO COMPRIMIR PROGRAMAS EN BASIC

Usted puede colocar más instrucciones y añadir potencia a sus programas en BASIC haciendo cada programa lo más corto posible. Ese proceso se llama comprimir. La compresión de programas le permite colocar el máximo número de instrucciones en su programa. También le ayuda a reducir el tamaño de los programas y ganar velocidad; y si usted escribe un programa que requiera una gran entrada de datos, como por ejemplo un inventario, un programa corto deja más memoria para colocar los datos.

PALABRAS ABREVIADAS

En el apéndice A se muestra una lista de abreviaciones de las palabras reservadas BASIC. Esto es una ayuda cuando escriba programas, ya que puede colocar más información en cada línea usando abreviaciones. La más frecuente es el signo

de interrogación (?), que es la abreviación BASIC de la instrucción PRINT. Sin embargo, si lista un programa que contenga abreviaciones, el Commodore 64 le mostrará todas las palabras completas. Si una línea de programa excede los 80 caracteres (2 líneas de pantalla) con las palabras sin abreviar y desea cambiar algo en ella, deberá entrar la línea con las palabras abreviadas. Guarde sus programas sin exceder la longitud de las líneas. Normalmente, las abreviaciones se añaden cuando el programa está escrito, y no deba listarse antes de guardarlo (SAVE).

REDUZCA LOS NUMEROS DE LINEA DEL PROGRAMA

Muchos programadores inician sus programas en la línea 100 y las numeran en intervalos de diez (100,110,120,etc.). Esto permite intercalar nuevas líneas de programa (111,112,etc.) a medida que el programa se desarrolla. Una de las formas de comprimir un programa cuando ya está terminado es cambiar los números de línea por los más cortos posibles (1,2,3,etc.), ya que los números de línea largos ocupan más memoria que los cortos. Por ejemplo: el número 100 utiliza 3 bytes de memoria (uno por cada cifra), mientras que el número 1 sólo ocupa 1 byte.

UTILICE LINEAS CON MULTIPLES INSTRUCCIONES

Usted puede colocar más de una instrucción en una línea de programa separándolas mediante dos puntos (:). La única limitación es que el total de instrucciones, incluyendo los dos puntos, no puede exceder de 80 caracteres. A continuación se muestra como ejemplo un programa antes y después de comprimirlo:

ANTES DE COMPRIMIR

```
10 PRINT "HOLA...";
20 FOR T=1 TO 500:NEXT
30 PRINT "HOLA, OTRA VEZ
..."
40 GOTO 10
```

DESPUES DE COMPRIMIR

```
10 PRINT "HOLA...";:FORT=1TO500:
NEXT:PRINT"HOLA, OTRA VEZ...":
GOTO10
```

QUITE LOS REM DE SUS PROGRAMAS

La instrucción REM es útil para recordar –o mostrar a otros programadores– que es lo que hace una sección especial de su programa. Sin embargo, cuando el programa está listo para su uso, probablemente no necesitará estos comentarios y quitándolos ahorrará una considerable cantidad de memoria. Si desea revisar o estudiar el programa en el futuro puede ser una buena idea guardar una copia archivada con las instrucciones REM intactas.

USE VARIABLES

Si un número o palabra se repite a menudo en un programa lo mejor es definirlo en una variable. Los números pueden definirse con una sola letra. Las palabras se definen como variables de cadena, y se debe usar una letra y el signo del dólar. He aquí un ejemplo.

ANTES DE COMPRIMIR

```
10 POKE 54296,15
20 POKE 54276,33
30 POKE 54273,10
40 POKE 54273,40
50 POKE 54273,70
60 POKE 54296,0
```

DESPUES DE COMPRIMIR

```
10 V54296:F54273
20 POKEV,15:POKE54276,33
30 POKEF,10:POKEF,40:POKEF,70
40 POKEV,0
```

USE LAS INSTRUCCIONES READ Y DATA

Gran cantidad de datos pueden tener que escribirse una y otra vez... o puede escribirlos UNA SOLA VEZ y almacenarlos en una lista mediante la instrucción DATA. Esto es especialmente bueno para colocar largas listas de números en un programa.

USE TABLAS Y MATRICES

Las tablas y matrices son similares a la instrucción DATA, ya que también se encargan de almacenar listas de datos. La diferencia entre las tablas (arrays) y las listas de DATA es que las primeras pueden ser multidimensionales.

ELIMINE ESPACIOS

Una de las formas más sencillas de comprimir un programa es quitando todos sus espacios. A pesar de que con frecuencia incluimos espacios en los programas de ejemplo para lograr mayor claridad, a estas alturas usted no necesita espacios en sus programas y ahorrará memoria si los elimina.

USE RUTINAS GOSUB

Si su programa debe ejecutar varias veces un conjunto de instrucciones, es mejor utilizar la instrucción GOSUB dirigida al inicio de esta serie de instrucciones, que escribirlas tantas veces como necesidad tenga el programa de ejecutarlas.

USE TAB Y SPC

En lugar de entrar varios comandos de movimiento del cursor para posicionar un carácter en un lugar determinado de la pantalla, es con frecuencia más acertado utilizar las instrucciones TAB y SPC para posicionar letras o palabras en la pantalla.

CAPITULO 2

VOCABULARIO DEL LENGUAJE BASIC

- Introducción
- Tipos de palabras, abreviaciones y funciones
- Descripción alfabética de palabras BASIC
- El teclado del Commodore 64 y sus características
- Editor de pantalla

INTRODUCCION

Este capítulo explica el significado de las palabras clave del BASIC CBM. En primer lugar se muestra una lista fácil de leer de las palabras, sus abreviaciones y cómo aparecen éstas en la pantalla. Después se explica con detalle la sintaxis y modo de trabajo de cada palabra reservada, mostrando ejemplos que le darán una idea de cómo utilizar la palabra en sus programas.
El BASIC del Commodore 64 le permite abreviar la mayoría de las palabras clave. Las abreviaciones se entran escribiendo la primera (o las dos primeras, según la palabra) letra de una palabra reservada, y después la segunda pulsando simultáneamente SHIFT.

Las abreviaciones NO AHORRAN memoria cuando se usan en programas, ya que cada palabra clave del BASIC es reducida a un solo carácter por el intérprete de BASIC. Cuando se lista un programa conteniendo abreviaciones, todas las palabras clave aparecerán completas, es decir, sin abreviar. Usted puede usar las abreviaciones para colocar más instrucciones en una línea, de forma que la línea –una vez vueltas las palabras a su escritura normal– puede ocupar más de 80 caracteres. Sin embargo, el Editor de Pantalla trabaja con líneas lógicas de 80 caracteres, por lo que las líneas así escritas no podrán ser modificadas. Para solventar esto hay dos soluciones: 1) Reescribir toda la línea, incluidas abreviaciones, o bien: 2) Dividir la línea excesivamente larga en dos, cada una con su propio número de línea.

Una lista completa de palabras reservadas y abreviaciones se muestra en la tabla 2-1. Esta tabla está seguida por una descripción alfabética de las instrucciones, comandos y funciones disponibles en su Commodore 64.
Este capítulo explica también todas las funciones BASIC incluidas en el intérprete. Estas funciones pueden ser usadas en modo directo o en programas, sin tener que definir la función previamente. Este NO es el caso de las funciones definibles por el usuario. Los resultados del uso de funciones se pueden utilizar para cualquier cálculo y se pueden asignar a variables del tipo apropiado. Hay dos tipos de funciones BASIC:

1) NUMERICAS
2) DE CADENA

Los argumentos en las funciones incluidas en el intérprete se encierran siempre entre paréntesis (). El paréntesis debe seguir inmediatamente a la función, NO PUDIENDOSE DEJAR ESPACIOS entre la última letra de la función y el paréntesis izquierdo (.

El tipo de argumento necesario se decide generalmente por el tipo de datos del resultado. Las funciones que dan como resultado una cadena se identifican por tener el signo del dolar ($) como último carácter de la palabra clave. En algunos casos las funciones de cadena pueden necesitar uno o varios argumentos numéricos.
Las funciones numéricas convierten el formato de los números de enteros a coma flotante si ello es necesario. En las descripciones que siguen, el tipo de datos del resultado se muestra junto al nombre de función. El tipo de argumentos se encuentra en el formato de la instrucción.

Tabla 2-1. PALABRAS CLAVE DEL BASIC DEL COMMODORE 64

COMANDO	ABREVIATURA	PANTALLA	TIPO DE FUNCION
ABS	A SHIFT B	A	NUMERICA
AND	A SHIFT N	A	
ASC	A SHIFT S	A	NUMERICA
ATN	A SHIFT T	A	NUMERICA
CHR$	C SHIFT H	C	DE CADENA
CLOSE	CL SHIFT O	CL	
CLR	C SHIFT L	C	
CMD	C SHIFT M	C	
CONT	C SHIFT O	C	
COS	ninguna	COS	NUMERICA
DATA	D SHIFT A	D	
DEF	D SHIFT E	D	
DIM	D SHIFT I	D	
END	E SHIFT N	E	
EXP	E SHIFT X	E	NUMERICA
FN	ninguna	FN	
FOR	F SHIFT O	F	
FRE	F SHIFT R	F	NUMERICA
GET	G SHIFT E	G	

COMANDO	ABREVIATURA	PANTALLA	TIPO DE FUNCION
GET#	none	GET#	
GOSUB	GO SHIFT S	GO ♥	
GOTO	G SHIFT O	G	
IF	ninguna	IF	
INPUT	ninguna	INPUT	
INPUT#	I SHIFT N	I	
INT	ninguna	INT	NUMERICA
LEFT$	LE SHIFT F	LE	DE CADENA
LEN	ninguna	LEN	NUMERICA
LET	L SHIFT E	L	
LIST	L SHIFT I	L	
LOAD	L SHIFT O	L	
LOG	ninguna	LOG	NUMERICA
MID$	M SHIFT I	M	DE CADENA
NEW	none	NEW	
NEXT	N SHIFT E	N	
NOT	N SHIFT O	N	
ON	none	ON	
OPEN	O SHIFT P	O	
OR	none	OR	

COMANDO	ABREVIATURA	PANTALLA	TIPO DE FUNCION
PEEK	P SHIFT E	P	NUMERICA
POKE	P SHIFT O	P	
POS	ninguna	POS	NUMERICA
PRINT	?	?	
PRINT#	P SHIFT R	P	
READ	R SHIFT E	R	
REM	ninguna	REM	
RESTORE	RE SHIFT S	RE ♥	
RETURN	RE SHIFT T	RE	
RIGHT$	R SHIFT I	R	DE CADENA
RND	R SHIFT N	R	NUMERICA
RUN	R SHIFT U	R	
SAVE	S SHIFT A	S ♠	
SGN	S SHIFT G	S	NUMERICA
SIN	S SHIFT I	S	NUMERICA
SPC(	S SHIFT P	S	DE CADENA
SQR	S SHIFT Q	S	NUMERICA
STATUS	ST	ST	NÚMERICA
STEP	ST SHIFT E	ST	
STOP	S SHIFT T	S	

COMANDO	ABREVIATURA	PANTALLA	TIPO DE FUNCION
STR$	ST SHIFT R	ST ⊟	DE CADENA
SYS	S SHIFT Y	S ◫	
TAB(	T SHIFT A	T ♠	DE CADENA
TAN	ninguna	TAN	NUMERICA
THEN	T SHIFT H	T ◫	
TIME	TI	TI	NUMERICA
TIME$	TI$	TI$	DE CADENA
TO	ninguna	TO	
USR	U SHIFT S	U ♥	NUMERICA
VAL	V SHIFT A	V ♠	NUMERICA
VERIFY	V SHIFT E	V ⊓	
WAIT	W SHIFT A	W ♠	

DESCRIPCION DE LAS PALABRAS CLAVE BASIC

ABS

TIPO:Función numérica
FORMATO:ABS(expresión)

Acción: Devuelve el valor absoluto del número, es decir, su valor sin signo. El valor absoluto de un número negativo es dicho número multiplicado por −1.

EJEMPLOS de la función ABS:

```
10 XABS(Y)
10 PRINT ABS(X*J)
10 IF XABS(X) THEN PRINT "POSITIVO"
```


AND

TIPO: Operador
FORMATO: expresiónANDexpresión

Acción: AND se usa en las operaciones Booleanas para tests de bits. **También se** usa para comprobar la verdad de ambos operandos.
En el álgebra Booleana, el resultado de una operación AND es 1 **sólo si los dos** números en los que se efectúa la operación son 1. **El resultado es 0 si uno o ambos** números son 0 (falso).

EJEMPLOS de la operacion AND de un bit:

```
    0        1        0        1
AND 0    AND 0    AND 1    AND 1
-----    -----    -----    -----
    0        0        0        1
```

El Commodore 64 realiza la operación AND en números comprendidos entre −32768 y 32767. Los valores fraccionarios no se usan, y los números fuera de rango producen el mensaje de error: **?ILLEGAL QUANTITY**

Cuando se convierte a formato binario, se permiten 16 bits para cada número. Los correspondientes bits son AND juntos, formando un resultado de 16 bits del mismo rango.

EJEMPLOS de la operacion AND de 16 Bits.

```
                      17
                 AND 194
------------------------
        0000000000010001
    AND 0000000011000010
------------------------
BINARIO 0000000000000000
------------------------
DECIMAL                0
```

```
                    -241
               AND 15359
------------------------
        1111111100001111
    AND 0011101111111111
------------------------
BINARIO 0011101100001111
------------------------
DECIMAL            15119
```

```
                   32007
               AND 28761
------------------------
        0111110100000111
    AND 0111000001011001
------------------------
BINARIO 0111000000000001
------------------------
DECIMAL            28673
```

Cuando se evalúa una comparación, se asigna un valor de -1 si el resultado es cierto, y un valor de cero si el resultado es falso. En formato binario, -1 es todo unos y 0 es todo ceros. Por esto, cuando se realizan evaluaciones cierto/falso mediante AND, el resultado es cierto si todos los bits en el resultado son ciertos.

EJEMPLOS del uso de AND en evaluaciones Verdadero/Falso:

```
50 IFX7 AND W3 THEN GOTO 10:REM SOLO VERDADERO SI X7 Y W3 ES
VERDADERO
60 IF A AND Q7 THEN GOTO 10:REM VERDADERO SI A0 Y Q7
```

ASC

TIPO: Función numérica
FORMATO: ASC(cadena)

Accion: ASC devuelve un número entre 0 y 255 que corresponde al valor ASCII de Commodore del primer carácter de la cadena. La tabla de dichos valores se muestra en el Apéndice C.

EJEMPLOS de la funcion ASC:

```
10PRINT ASC("Z")
20XASC("CEBRA")
30JASC(J$)
```

Si la cadena no contiene caracteres, aparecerá el mensaje de error ?ILLEGAL QUANTITY. En el tercer ejemplo, si J$"", la funcion ASC no trabajaría. Las instrucciones GET y GET# leen un CHR$(0) como cadena nula. Para eliminar el problema de la función ASC debe añadir CHR$(0) al final de la cadena como se muestra a continuación:

EJEMPLO de la función ASC evitando el error ILLEGAL QUANTITY:

```
30JASC(J$+CHR$(0))
```

ATN

TIPO: Función numérica
FORMATO: ATN (número)

Acción: Esta función matemática devuelve el arcotangente de un número. El resultado es el ángulo (en radianes) cuya tangente es el número dado. El resultado se encuentra siempre en el rango -/2 a /2.

EJEMPLOS de la funcion ATN:

```
10 PRINT ATN (0)
20 X ATN (J)* 180/ : REM CONVIERTE A GRADOS
```

CHR$

TIPO: Función de cadena
FORMATO: CHR$ (número)

Acción: Esta función convierte un código ASCII de Commodore al carácter equivalente. Vea en el Apéndice C la lista de caracteres y códigos. El número debe estar comprendido entre 0 y 255, de lo contrario se generará el mensaje de error ?ILLEGAL QUANTITY.

EJEMPLOS de la funcion CHR$:

```
10 PRINT CHR$(65) : REM 65 "A" MAYUSCULA
20 A$ CHR$(13) : REM 13 TECLA "RETURN"
30 A ASC(A$) : A$ CHR$(A): REM CONVERSION DE
   CARACTER A CODIGO Y VICEVERSA.
```

CLOSE

TIPO:Instrucción de E/S
FORMATO:CLOSE<número de fichero>

Acción: Esta instrucción cierra el canal de datos con un periférico. El número de fichero es el mismo de la instrucción OPEN que abrió el canal. (Vea la descripción de la instrucción OPEN y la sección de ENTRADAS/SALIDAS).

Cuando se trabaja con periférico de almacenamiento de datos como el cassette o la unidad de discos, la instrucción CLOSE almacena los buffers incompletos en el periférico. Si la instrucción no se ejecuta, el fichero quedara incompleto en el cassette o ilegible en el diskette. La instrucción CLOSE no es necesaria con otros periféricos, pero libera memoria para otros ficheros. Vea el manual de su periférico para más detalles.

EJEMPLOS de la instrucción CLOSE

```
10 CLOSE1
20 CLOSEX
30 CLOSE9*(1+J)
```

CLR

TIPO:Instrucción
FORMATO:CLR

Acción: Esta instrucción repone RAM usada cuya información no se vaya a necesitar en lo sucesivo. Los programas BASIC en memoria no son alterados, pero todas las variables, tablas, direcciones de retorno de GOSUB, bucles FOR...NEXT, funciones definidas por el usuario, y ficheros son borrados de la memoria, y el espacio que han dejado queda disponible para nuevas variables, etc.

En el caso de ficheros en cassette o disco, los mismos no se cierran adecuadamente mediante la instrucción CLR. La información acerca de los ficheros que se almacena en el ordenador se pierde, incluyendo el contenido de los buffers incompletos. La unidad de disco cree que el fichero sigue abierto. Vea la instrucción CLOSE para más información acerca de esto.

EJEMPLO de la instrucción CLR:

```
10X=25
20CLR
30SPRINT X

RUN
0

READY
```

CMD

TIPO: Instrucción de E/S
FORMATO: CMD<número de fichero>[,cadena]

Acción: Esta instrucción cambia el periférico de salida normal (T.V.) al periférico especificado. Este puede ser el cassette, el diskette, la impresora o cualquier periférico de E/S como el modem. El número de fichero debe ser especificado en una orden OPEN anterior. La cadena, si se incluye, es enviada al periférico. Esta característica es útil para imprimir títulos en los listados de programas.
Cuando se ejecuta este comando, cualquier instrucción PRINT, y el comando LIST no se muestran en la pantalla, pero se envía el texto en el mismo formato al periférico.
Para redireccionar la salida a la pantalla, debe enviarse un PRINT# con una línea en blanco antes de cerrar el periférico, para que el mismo deje de esperar datos. Cualquier error del sistema (como ?SINTAX ERROR) produce el retorno de la salida a la pantalla. Después de aparecido el mensaje se debe enviar una línea en

blanco al periférico para que deje de esperar datos. (Vea el manual de la impresora o el diskette para más detalles.)

EJEMPLOS de la instrucción CMD:

```
OPEN 4,4:CMD 4,"TITULO":LIST:REM LISTA UN PROGRAMA EN LA
IMPRESORA PRINT#4:CLOSE4:CIERRA EL CANAL DE LA IMPRESORA.


10OPEN1,1,1,"TEST":REM CREA FICHERO SECUENCIAL
20CMD1:REM SALIDA AL CASSETTE EN LUGAR DE LA PANTALLA
30FOR L=1 TO 100
40PRINT L:REM COLOCA EL NUMERO EN EL BUFFER DE CASSETTE
50NEXT
60PRINT#1:CLOSE1:REM ESCRIBE EN EL CASSETTE EL BUFFER
INCOMPLETO, CIERRA EL CANAL CORRECTAMENTE.
```

CONT

TIPO:Comando
FORMATO:CONT

Acción: Este comando reinicia un programa detenido por la instrucción STOP o END o por la pulsación de la tecla RUN/STOP. El programa continúa justo en el lugar en que se detuvo.
Mientras el programa está detenido, el usuario puede inspeccionar el valor de las variables y mirar el listado del programa. Cuando se depura un programa es útil colocar instrucciones STOP en lugares estratégicos para poder examinar las variables y ver el curso del programa.
Si intenta editar un programa parado (incluso pulsando la tecla RETURN en una línea no cambiada) aparecerá el mensaje de error **CAN'T CONTINUE**. Esto sucederá también si el programa se ha detenido por un error, o si ha causado un error antes de entrar CONT.

EJEMPLO del comando CONT:

```
10PI=0:C=1
20PI=PI+4/C-4/(C+2)
30PRINT PI
40C=C+4:GOTO20
```

Este programa calcula el valor de PI. Ejecútelo y al cabo de un rato pulse la tecla RUN/STOP. Usted podrá ver el siguiente mensaje:

BREAK IN 20 (NOTA: Puede aparecer otro número.)

Escriba el comando PRINT C para ver lo lejos que el ordenador ha llegado en sus calculos. Después use CONT para proseguir la ejecución a partir del punto en que el programa se detuvo.

COS

TIPO: Función
FORMATO: COS (<número>)

Acción: Esta función matemática calcula el coseno del número, donde el numero es un ángulo en radianes.

EJEMPLOS de la función COS:

```
10 PRINT COS(0)
20 X=COS(Y*π/180):REM CONVIERTE GRADOS A RADIANES)
```

DATA

TIPO: Instrucción
FORMATO: DATA <lista de constantes>

Acción: Las instrucciones DATA almacenan información dentro del programa. El mismo utiliza esta información mediante el comando READ, que extrae las constantes sucesivamente de las instrucciones DATA.
Las instrucciones data no son ejecutadas por el ordeñador. Unicamente están presentes en el programa. Por esto, normalmente se sitúan al final del programa. Todas las instrucciones DATA de un programa son tratadas como una lista continua. Los datos son leídos de izquierda a derecha, desde el número de línea menor al mayor. Si una instruccion READ encuentra un dato de tipo distinto al requerido (si necesita un número y encuentra una cadena), aparece un mensaje de error y el programa se detiene.
Se puede incluir cualquier caracter en las listas DATA, pero algunos deben encerrarse entre comillas (""). Esta excepción incluye a los signos de puntuación como coma (,), dos puntos (:), espacios en blanco, letras entradas con SHIFT, gráficos y caracteres de control del cursor o color.

EJEMPLOS de la instrucción DATA:

```
10 DATA 1,10,5,8
20 DATA JUAN PABLO, JORGE
30 DATA "QUERIDA MARY, COMO ESTAS, TE QUIERE, BILL"
40 DATA -1.7E,-9,3.33
```


DEF FN

TIPO: Instrucción
FORMATO: DEF FN <nombre> (<variable>)=<expresión>

Acción: Esta instrucción activa una función definida por el usuario que puede ser usada mas tarde por el programa. La función puede consistir en cualquier fórmula matematica. Las funciones definidas por el usuario ahorran memoria cuando una formula larga se debe usar en varias partes de un programa. La fórmula solo se debe especificar una vez, en la instrucción de definición, y entonces se abrevia a un nombre de función. El nombre de función consiste en las letras FN seguidas de cualquier nombre de variable. Esta variable puede tener 1 o 2 caracteres, el primero debe ser una letra, y el segundo puede ser una letra o un número.

EJEMPLOS de la instrucción DEF FN:

```
10DEF FN A(X)=X+7
20DEF FN AA(X)=Y*Z
30DEF FN A9(Q)=INT <RND(1)*Q+1>
```

La función se llama en el programa mediante el nombre de función seguido de una variable entre paréntesis. El nombre de función se usa igual que cualquier variable, y su valor es calculado automáticamente.

EJEMPLOS del uso de FN

```
40 PRINT FN A(9)
50 R=FNAA(9)
60 G=G-FN A9(10)
```

En la línea 50 de los ejemplos, el número nueve entre paréntesis no afecta al resultado de la función, porque en la definición de la función realizada en la linea 20 no se usa la variable entre paréntesis. El resultado es Y veces Z, indiferentemente del valor de X. En las otras dos funciones, el valor entre paréntesis afecta al resultado.

DIM

TIPO: Instrucción
FORMATO: DIM <variable> (<suscritos>) Ç,<variable> (<suscritos>)...ç

Acción: Esta instrucción define una tabla o matriz de variables (array). Esto le permite usar el nombre de variable con un suscrito. El suscrito indica el elemento deseado. El número de elemento mas bajo en una tabla es 0, y el myor se proporciona en la instrucción DIM, con un máximo de 32767.

La instrucción DIM se debe ejecutar una y solo una vez por cada tabla. Si la instrucción se reejecuta aparecerá el mensaje de error REDIM'D ARRAY. Es por esto que en la mayoría de los programas las instrucciones DIM se encuentran al principio de los mismos.
Una tabla puede tener cualquier número de dimensiones, limitándose solo por la memoria disponible para almacenar las variables. La tabla se puede utilizar para almacenar variables numéricas normales o bien variables enteras o de cadena. En este caso coloque detras del nombre de variable los signos % y $, respectivamente.
Si una tabla no se ha dimensionado previamente en un programa, se dimensiona automáticamente a 11 elementos en cada dimensión usada en la primera referencia a la misma.

EJEMPLOS de la instrucción DIM:

```
10DIM A(100)
20DIM Z(5,7),Y(3,4,5)
30DIM Y7%(Q)
40DIM PH$(1000)
50F(4)=9:REM REALIZA AUTOMATICAMENTE DIM F(10)
```

EJEMPLO de TABLERO DE PUNTUACION DE FUTBOL usando DIM:

```
10DIM S(1,5),T$(1)
20INPUT NOMBRE EQUIPOS";T$(0),T$(1)
30FOR Q=1 TO 5:FOR T=0 TO 1
40PRINT T$(T),"PUNTUACION EN EL TIEMPO"Q
50INPUT S(T,Q):S(T,0);S(T,0)+S(T,Q)
60NEXT T,Q
70PRINT CHR$(147)"TABLA DE PUNTUACION"
80PRINT "TIEMPO"
90FOR Q=1 TO 5
100PRINT TAB(Q*2+9)Q;
110NEXT: PRINT TAB(15)"TOTAL"
120FOR T=0 TO 1:PRINT T$(T)
130FOR Q=1 TO 5
140PRINT TAB(Q*2+9)S(T,Q)
150NEXT:PRINT TAB(15)S(T,0)
160NEXT
```

CALCULO DE LA MEMORIA CONSUMIDA POR DIM:

5 bytes por el nombre de tabla
2 bytes por cada dimensión
2 bytes/elemento en variables enteras
5 bytes/elemento en variables numéricas normales
3 bytes/elemento en variables de cadena +
1 byte por caracter en cada variable de cadena

END

TIPO:Instrucción
FORMATO:END

Acción: Esta instrución termina la ejecución de un programa e imprime el mensaje READY, devolviendo el control al usuario. Se puede colocar cualquier número de instrucciones END en un programa. La instrucción END no es necesaria si el programa termina en la última linea escrita, pero si el programa debe terminar en algun punto intermedio debe colocarse esta instrucción para que él mismo pare. La instrucción END funciona igual que la STOP. La única diferencia es que STOP produce el mensaje **BREAK IN LINE XX** y END solo muestra READY. Ambas instrucciones permiten al ordenador reemprender el programa mediante el comando CONT.

EJEMPLOS de la instrucción END:

```
10 PRINT"DESEA UTILIZAR ESTE PROGRAMA"
20 INPUT A$
30 IF A$="NO"THEN END
40 REM RESTO DEL PROGRAMA...
999 END
```

EXP

TIPO:Función-Numérica
FORMATO:EXP(<número>)

Acción: Esta función matemática calcula la constante e (2.71828183) elevada a la potencia del número dado. Un valor mayor de 88.0296919 causa el error ?OVERFLOW.

EJEMPLOS de la función EXP:

```
10 PRINT EXP(1)
20 X=Y*EXP(Z*Q)
```

FN

TIPO:Función-Numérica
FORMATO:FN<nombre>(<número>)

Acción: Esta función hace referencia a la fórmula del mismo nombre DEFinida anteriormente. El numero es colocado en su lugar en la fórmula (si es necesario) y la fórmula es calculada. El resultado es un valor numérico.

Esta función puede ser usada en modo directo, siempre que se haya ejecutado anteriormente la instrucción DEF.
Si se utiliza FN antes de DEFinir la función aparecerá el mensaje de error **UNDEF'D FUNCTION.**

EJEMPLOS de la función FN:

```
PRINT FN A(Q)
1100 J=FN J(7)+FN J(9)
9900 IF FN B7(I+1)=6 THEN END
```

FOR...TO...STEP

TIPO: Instrucción
FORMATO:FOR<variable>=<inicio>TO<limite> [STEP<incremento>]

Acción: Esta es una instrucción especial del BASIC que le permite usar con facilidad una variable como contador. Usted debe especificar varios parámetros: el nombre de variable de coma flotante, su valor de inicio, el límite de la cuenta, y -opcionalmente- el incremento por cada ciclo.
A continuación se muestra un pequeño programa que cuenta de 1 a 10, imprime cada número y finaliza al terminar la cuenta. No se ha utilizado la instrucción FOR:

```
100 L=1
110 PRINT L
120 L=L+1
130 IF L<=10 THEN 110
140 END
```

Usando la instrucción FOR, el programa queda así:

```
100 FOR L=1 TO 10
110 PRINT L
120 NEXT L
130 END
```

Como puede ver, el programa realizado de esta forma es más corto y fácil de entender.
Varias operaciones se realizan durante la ejecución de la instrucción FOR. El valor de <inicio> se asigna a la <variable> usada como contador. En el ejemplo anterior, se coloca el valor 1 en la variable L.
Cuando se ejecuta la instrucción NEXT, el valor de <incremento> se suma a la <variable>. Si no se incluye STEP, el incremento se ajusta a +1. La primera vez que

el programa anterior ejecuta la instrucción 120, se suma 1 a L, con lo que L tendra el valor de 2.
Ahora el valor de la variable es comparado con el <limite>. Si todavia no se ha alcanzado este, el programa ejecutará la instrucción siguiente a la FOR. En este caso, como el valor de L (2) es inferior al límite (10), el programa ejecutará la linea 110.
Llega un momento en que el valor de la variable excede el límite. En este punto el programa ejecuta la línea siguiente a la NEXT. En nuestro ejemplo, cuando L alcanza el valor 11, que excede a 10, el programa ejecutará la linea 130.
Cuando el valor del incremento es positivo la variable debe exceder el límite para que el bucle concluya. Cuando el incremento es negativo la variable debe ser menor que el incremento para que esto ocurra.

> **NOTA:** Un bucle se ejecuta siempre como mínimo una vez.

EJEMPLOS de la instrucción FOR...TO...NEXT.. STEP

```
100 FOR L-100 TO 0 STEP -1
100 FOR L-PI TO 6*X STEP .01
100 FOR AA- 3 TO 3
```

FRE

TIPO:Función
FORMATO:FRE(<variable>)

Acción: Esta función le indica la cantidad de memoria RAM disponible para programas y variables. Si un programa intenta ocupar más memoria de la disponible aparecera el mensaje de error **OUT OF MEMORY.**
El número entre paréntesis puede tener cualquier valor, no siendo utilizado en el cálculo.

> **NOTA:** Si el resultado de FRE es negativo, sume 65536 al resultado de FRE para obtener el numero de bytes disponibles.

EJEMPLOS de la función FRE:

```
PRINT FRE(0)
10 X=(FRE(0) 1000)/7
95 IF FRE(0)<100 THEN PRINT"QUEDA POCA MEMORIA"
```

> **NOTA:** La siguiente fórmula le indica siempre la memoria RAM disponible:

```
PRINT FRE(0)-(FRE(0)<0)*65536
```

GET

TIPO:Instrucción
FORMATO:GET<lista de variables>

Acción: Esta instrucción lee cada tecla pulsada por el usuario. Cuando el usuario escribe, los caracteres se almacenan en el buffer de teclado presente en el Commodore 64. Este buffer tiene capacidad para 10 caracteres, y las teclas pulsadas después del decimo caracter se pierden. Leyendo uno de estos caracteres mediante GET se deja espacio para que otro pueda entrar en el buffer.
Si la instrucción GET especifica datos numéricos, y el usuario pulsa una tecla no numérica, aparece el mensaje de error **?SYNTAX ERROR**. Para su seguridad, lea el teclado como cadenas y convierta posteriormente las mismas a números.
La instrucción GET se puede usar para superar ciertas limitaciones de la instrucción INPUT. Para más detalles sobre esto consulte el capítulo sobre el uso de la instrucción GET en la sección sobre técnicas de programación.

EJEMPLOS de la instrucción GET:

```
10  GET A$:IF A$=""THEN 10:REM DETIENE LA EJECUCION HASTA
    QUE SE PULSA UNA TECLA
20  GET A$,B$,C$,D$,E$:REM LEE CINCO TECLAS
30  GET A,A$
```

GET#

TIPO:Instrucción de E/S
FORMATO:GET# <número de fichero>, <lista de variables>

Acción: Esta instrucción lee caracteres de uno en uno del periférico especificado. Trabaja del mismo modo que GET, excepto que los datos proceden de un periférico en lugar del teclado. Si no se recibe ningún carácter, la variable contiene una cadena vacía (igual a "") o un 0 si es variable numérica. Los caracteres para separar datos en los ficheros como coma (,) o el código de la tecla RETURN (código 13 ASC), son recibidos como cualquier otro carácter.
Cuando se usa con el periférico #3 (pantalla), esta instrucción lee caracteres de la pantalla uno a uno. Cada ejecución de GET# mueve el cursor un carácter a la derecha. El carácter del final de línea lógica es convertido a CHR$(13), el código de la tecla RETURN.

EJEMPLOS de la instrucción GET#:

```
5   GET#1,A$
10  OPEN 1,3:GET#1,Z7$
20  GET#1,A,B,C$,D$
```


GOSUB

TIPO:Instrucción
FORMATO:GOSUB<número de línea>

Acción: Esta es una forma especial de GOTO, con una diferencia importante: GOSUB recuerda de la línea de donde ha venido. Cuando se encuentra en el programa la instrucción RETURN (distinta de la tecla RETURN), el programa regresa a la instrucción siguiente a la GOSUB.
El uso más frecuente de una sub-rutina –GOSUB significa GO to SUBroutine (vea sub-rutina)– es cuando una pequeña parte del programa se debe ejecutar varias veces por distintas secciones del programa. Usando subrutinas en lugar de repetir las mismas líneas una y otra vez en distintas partes del programa puede ahorrar gran cantidad de memoria. En cierto modo, GOSUB es similar a DEF FN. DEF FN le permite ahorrar espacio cuando usa una fórmula, y GOSUB ahorra espacio cuando se utiliza una rutina de varias líneas. A continuación se muestra un programa ineficiente al carecer de instrucciones **GOSUB**:

```
100 PRINT "ESTE PROGRAMA IMPRIME"
110 FOR L=1 TO 500:NEXT
120 PRINT "LENTAMENTE EN LA PANTALLA"
130 FOR L=1 TO 500:NEXT
140 PRINT "USANDO UN BUCLE"
150 FOR L=1 TO 500:NEXT
160 PRINT "COMO RETARDO"
170 FOR L=1 TO 500:NEXT
```

He aquí el mismo programa usando GOSUB:

```
100 PRINT "ESTE PROGRAMA IMPRIME"
110 GOSUB 200
120 PRINT "LENTAMENTE EN LA PANTALLA"
130 GOSUB 200
140 PRINT "USANDO UN BUCLE"
150 GOSUB 200
160 PRINT "COMO RETARDO"
170 GOSUB 200
180 END
200 FOR L=1 TO 500:NEXT
210 RETURN
```

Cada vez que el programa ejecuta una instrucción GOSUB, el número de línea y posición en el programa se almacena en una zona especial de memoria llamada "stack", que ocupa 256 bytes de memoria. Esta pequeña cantidad de memoria limita la capacidad del stack, por esto, el número de direcciones de retorno que se pueden almacenar es limitado, y debe asegurarse de que cada GOSUB tenga su correspondiente RETURN, o podría encontrarse con error de falta de memoria a pesar de disponer de mucha RAM libre para su programa.

GOTO

TIPO:Instrucción
FORMATO:GOTO <número de línea>
o GO TO <número de línea>

Acción: Esta instrucción permite a un programa en BASIC ejecutar líneas sin seguir el número de orden ascendente. La instrucción GOTO seguida de un número de línea obliga al programa a proseguir la ejecución a partir de la línea especificada. GOTO NO seguido por ningun número equivale a GOTO 0 Detrás de GOTO se debe colocar un número. No está permitida la colocación de una variable.
Es posible crear bucles sin fin mediante la instrucción GOTO. El caso más usual es una línea que se dirija a ella misma, como 10 GOTO 10. Estos bucles pueden ser detenidos usando la tecla RUN/STOP.

EJEMPLOS de la instrucción GOTO:

```
GOTO 100
10 GO TO 50
20 GOTO 999
```

IF...THEN...

TIPO:Instrucción
FORMATO:IF<expresión>THEN<número de línea>
IF<expresión>GOTO<número de línea>
IF<expresión>THEN<instrucciones>

Acción: Esta es una de las instrucciones que da al BASIC la mayor parte de su "inteligencia", permite evaluar condiciones y realizar distintas acciones dependiendo del resultado de la condición.
La palabra IF es seguida por una expresión, que puede incluir variables, cadenas, numeros, comparaciones y operadores lógicos. La palabra THEN debe colocarse en la misma linea y es seguida por un numero de linea o por más instrucciones BASIC Si la expresión es falsa, todo lo que se encuentre detrás de THEN en la linea es ignorado, y la ejecución prosigue en la siguiente línea del programa. Un resultado cierto produce que el programa salte al número de linea especificado despues de THEN o ejecute las instrucciones que siguen a esta palabra.

EJEMPLO de la instrucción IF...GOTO...

```
100 INPUT "ESCRIBA UN NUMERO";N
110 IF N<=0 GOTO 200
120 PRINT "RAIZ CUADRADA="SQR(N)
130 GOTO 100
200 PRINT "EL NUMERO DEBE SER >0"
210 GOTO 100
```


Este programa calcula e imprime la raíz cuadrada de cualquier número positivo. La instrucción IF se usa para validar el numero entrado mediante INPUT. Cuando el resultado de N<=0 es cierto, el programa salta a la línea 200, y cuando es falso la próxima línea a ejecutar es la 120. Advierta que no es necesario colocar THEN GOTO con el IF...THEN en la línea 110. GOTO 200 significa en este caso THEN GOTO 200.

EJEMPLO de IF...THEN...

```
100 FOR L=1 TO 100
110 IF RND(1)<.5 THEN X=X+1:GOTO 130
120 Y=Y+1
130 NEXT
140 PRINT "MENORES DE .5"X
150 PRINT "MAYORES DE .5"Y
```

El IF en la línea 110 comprueba si el número aleatorio es menor de .5. Si el resultado es cierto, se ejecutan las instrucciones que siguen a THEN: primero se incrementa X en una unidad, después el programa salta a la línea 130. Cuando el resultado es falso, el programa pasa a la próxima línea, en este caso la 120.

INPUT

TIPO:Instrucción
FORMATO:INPUT["<mensaje>"];<lista de variables>

Acción: Esta es una instrucción que permite al usuario del programa entrar información en el ordenador. Cuando se ejecuta esta instrucción se imprime un interrogante (?) en la pantalla, y el cursor se posiciona un espacio a la derecha del interrogante. Ahora el ordenador espera, con el cursor parpadeando, que el usuario escriba los datos y pulse la tecla RETURN.
La palabra INPUT puede ser seguida por cualquier texto entre comillas (""). Este texto se imprime en la pantalla, seguido del signo de interrogación.
Después del texto debe colocarse punto y coma (;) y el nombre de una o varias variables, separadas por comas. El ordenador almacena los datos entrados en la variable especificada, debiendo utilizar variables distintas para cada entrada.

EJEMPLOS de la instrucción INPUT:

```
100 INPUT A
110 INPUT B,C,D
120 INPUT "PREGUNTA";E
```

Cuando se ejecute este programa, aparecerá el interrogante para avisar al operador de que el ordenador espera la entrada de datos. Cualquier número entrado se asigna a la variable A, que puede ser usada posteriormente por el programa. Si la respuesta no es un numero, aparece el mensaje **?REDO FROM START**, lo que significa que se ha entrado una cadena y el ordenador esperaba un número. Si el operador sólo pulsa RETURN, el valor de la variable no cambia.

Ahora aparece el siguiente interrogante (línea 110). Si sólo se entra un número, el Commodore 64 mostrará 2 interrogantes (??), lo que significa que el ordenador requiere más datos.
Usted puede entrar de una vez más de un dato, que deberá separar mediante comas. De este modo previene la aparición del doble signo de interrogación. Si escribe más datos de los necesarios aparecerá el mensaje **?EXTRA IGNORED**, lo que significa que los datos extra que escribió no se asignan a ninguna variable.
La línea 120 muestra la palabra PREGUNTA antes de que aparezca el interrogante. El punto y coma es necesario entre la pregunta y la lista de variables.
La instrucción INPUT no se puede usar nunca fuera de programa. El Commodore 64 precisa espacio para un buffer que almacene temporalmente la respuesta a un INPUT, y este espacio es el mismo que se reserva para comandos en modo directo.

INPUT#

TIPO:Instrucción de E/S
FORMATO:INPUT# <número de fichero>, <lista de variables>

Acción: Esta es normalmente la vía más fácil y rápida de extraer datos almacenados en disco o cassette. Los datos se extraen por variables completas, siempre que éstas tengan hasta 80 caracteres, a diferencia de la instrucción GET#, que los extrae letra a letra. Para poder utilizar INPUT# es necesario abrir antes el fichero mediante OPEN.
La instrucción INPUT# asume que una variable ha terminado cuando lee el código de RETURN (CHR$(13)), una coma (,), punto y coma (;), o dos puntos (:). Deben usarse comillas ("") cuando se escriben estos símbolos como parte de una variable. (Vea la instrucción PRINT#)
Si la variable usada es numérica y se reciben caracteres no numéricos se produce el error **BAD DATA**. INPUT# puede leer cadenas de hasta 80 caracteres. El intento de leer una cadena mayor dará como resultado el error **STRING TOO LONG**. Cuando se usa el periférico #3 (la pantalla), esta instrucción lee una línea lógica entera y mueve el cursor una línea hacia abajo hasta la próxima línea.

EJEMPLOS de la Instruccion INPUT#:

```
10 INPUT#1,A
20 INPUT#2,A$,B$
```

INT

TIPO:Función Entera
FORMATO:INT(<número>)

Acción: Devuelve el valor entero de la expresión. Si la expresión es positiva, se ignora la parte decimal. Si la expresión es negativa, cualquier decimal causa que se devuelva el entero más bajo

EJEMPLOS de la función INT:

```
120 PRINT INT(99.4343),INT(-12.34)
 99                  -13
```

LEFT$

TIPO:Función de cadena
FORMATO:LEFT$(<cadena>,<entero>)

Acción: Devuelve una cadena que comprende el número de caracteres más a la izquierda de la cadena determinados por el entero de la función. El argumento entero puede ir de 0 a 255. Si el entero es mayor que la longitud total de la cadena, el resultado es la cadena entera. Si se usa cero como entero, el resultado es una cadena vacía (de cero caracteres).

EJEMPLOS de la función LEFT$:

```
10 A="COMMODORE COMPUTERS"
20 B=LEFT$(A$,9):PRINT B$
RUN

COMMODORE
```

LEN

TIPO:Función entera
FORMATO:LEN(<cadena>)

Acción: Devuelve el número de caracteres de la cadena. Los caracteres de control y espacios también se incluyen en la cuenta.

EJEMPLO de la función LEN:

```
CC$="COMMODORE COMPUTER":PRINT LEN(CC$)

18
```

LET

TIPO:Instrucción
FORMATO:[LET]<variable>=<expresión>

Acción: La instrucción LET puede ser usada para asignar un valor a una variable. Pero como que LET es opcional, la mayoría de programadores lo omiten para

ahorrar tiempo y memoria. El signo igual (=) es suficiente cuando se asigna el valor de una expresión a una variable.

EJEMPLOS de la instrucción LET:

```
10 LET D=12                 (Esta línea es igual a: 10 D=12)
20 LET E$="ABC"
30 F$="DEF"
40 LET SUM$=E$+F$           (SUM$ es igual a ABCDEF)
```

LIST

TIPO:Comando
FORMATO:LIST[<primera línea>]-[<última línea>]

Acción: El comando LIST le permite ver las líneas del programa BASIC en la memoria de su Commodore 64. Esta característica le permite usar las facilidades del potente editor de pantalla de su ordenador con el fin de editar y corregir las líneas que se han mostrado mediante LIST con suma facilidad.
El comando LIST muestra todo o parte de un programa residente en memoria al periférico asignado. LIST se dirige normalmente a la pantalla, pero sin embargo, utilizando la instrucción CMD se puede enviar a otro periférico como la impresora o el disco. El comando LIST puede ejecutarse en un programa, pero después del listado aparecerá el mensaje READY, deteniéndose el programa.
Cuando lista un programa en pantalla, la velocidad de desplazamiento de las líneas (SCROLL) puede ser reducida pulsando la tecla de ConTRoL CTRL. El listado es interrumpido pulsando la tecla RUN/STOP.
Si no se especifican números de línea se listará el programa por entero. Si se especifica un número de línea seguido de un guión (–), la línea mencionada y todas las superiores serán mostradas. Si se coloca delante el guión y detrás un número de línea se mostrarán todas las líneas inferiores al número, y también la correspondiente al número escrito. Si se especifican los dos números de línea se mostrarán todas las comprendidas entre ellos, ambos inclusive.

EJEMPLOS del comando LIST:

```
LIST  (Lista el programa en memoria)
LIST 500  (Lista la línea 500)
LIST 150-  (Lista desde la línea 150 al final)
LIST -1000  (Lista desde el principio a la línea 1000)
LIST 150-1000  (Lista de la línea 150 a la 1000, inclusive)
```

```
10 PRINT "ESTA ES LA LINEA 10"
20 LIST (LIST usado en modo programa)
30 PRINT "ESTA ES LA LINEA 30"
```


LOAD

TIPO:Comando
**FORMATO:LOAD["<nombre fichero>"][,<periférico>]
[<dirección>]**

Acción: El comando LOAD lee un fichero de programa desde el cassette o la unidad de disco, colocándolo en memoria. De esta forma usted puede usar la información cargada cambiarla de cualquier modo. El número de periférico es opcional, pero si no se especifica el ordenador lo considera como 1, número que corresponde a la unidad de cassette. El número de la unidad de disco es normalmente 8. El comando LOAD cierra todos los ficheros abiertos y, si se usa en modo directo, realiza un CLR (borrar variables) antes de cargar el programa. Si LOAD se ejecuta dentro de un programa, una vez cargado se realizará un RUN, sin la pérdida de ninguna de las variables. De esta forma puede "encadenar" varios programas juntos.
Usando la unidad de disco y colocando como nombre de fichero un asterisco (*), el ordenador cargará el primer programa que figure en el directorio. Si el nombre de fichero especificado no existe o no corresponde a un fichero de programa se generará el error **?FILE NOT FOUND**.
Cuando se cargan programas desde el cassette, el nombre de fichero puede ser omitido, cargándose el primer programa que se encuentre en la cinta. El Commodore 64 colocará la pantalla del color del borde una vez se haya pulsado la tecla PLAY. Cuando se encuentra el programa, la pantalla retorna a la normalidad y aparece el mensaje "FOUND". Al pulsar la tecla C= o después de una pausa de aproximadamente 15 segundos, el programa se cargara en el ordenador. Si se pulsa la barra de espacio, el ordenador ignorará el programa que acaba de encontrar para intentar cargar el siguiente. Los programas se cargan a partir de la posición 2048 a menos que se use una dirección secundaria 1. Si usa la dirección secundaria 1 el programa se cargará en la misma zona de memoria donde se encontraba al ser almacenado (SAVE).

EJEMPLOS del comando LOAD:

LOAD	(Lee el próximo programa del cassette)
LOAD A$	(Busca el programa cuyo nombre esta contenido en A$)
LOAD"*" ,8	(Carga el primer programa del disco)
LOAD"",1,1	(Busca el Primer Programa en el cassette y lo carga en el mismo lugar de la memoria en que estaba cuando se grabó.

```
LOAD"STAR TREK"   (Carga un programa desde cassette)
PRESS PLAY ON TAPE
FOUND STAR TREK
LOADING
READY

LOAD "FUN" ,8      (Carga un programa desde disco)
SEARCHING FOR FUN
LOADING
READY

LOAD "JUEGO" ,8,1  (Carga un programa a partir de la posición
                   de memoria en que fue guardado anteriormente
                   en el disco)
```

LOG

TIPO:Función de coma flotante
FORMATO:LOG(<numérico>)

Acción: Da como resultado el logaritmo natural (log en la base e) del argumento. Si el valor del argumento es cero o negativo se genera el mensaje de error **?ILLEGAL QUANTITY.**

EJEMPLOS de la función LOG:

```
25 PRINT LOG(45/7)
 1.86075234

10 NUM=LOG(ARG)/LOG(10) (Calcula el LOG de ARG en base 10).
```

MID$

TIPO:Función de cadena
FORMATO:MID$(<cadena>,<numérico-1> [<numérico-2>])

Acción: La función MID$ devuelve una subcadena de la especificada en <cadena>. La posición de inicio de la subcadena se define por <numérico-1> y la longitud de la misma por <numérico-2>. Ambos argumentos deben estar comprendidos entre 0 y 255.
Si el valor <numérico-1> es mayor que la longitud de la cadena o si el valor <numérico-2> es cero, la funcion MID$ dará como resultado una cadena vacia. Si el valor <numérico-2> está fuera de rango, el ordenador asume que debe usarse el resto de la longitud de la cadena.

EJEMPLO de la función MID$:

```
10 A$="HOLA "
20 B$="PEDRO JOSE ALBERTO"
30 PRINT A$+MID$(B$,6,4)

HOLA JOSE
```

NEW

TIPO:Comando
FORMATO:NEW

Acción: El comando NEW se usa para borrar el programa actualmente en memoria, asi como todas las variables. Antes de escribir un nuevo programa es conveniente usar NEW para borrar la memoria del ordenador. NEW también se puede usar en un programa, pero debe tener mucho cuidado al colocar este comando en el mismo, ya que accidentalmente podría perder el programa en memoria, por ejemplo, al corregir los errores del mismo.

> **ATENCION:** No borrar un programa residente en memoria cuando se va a escribir uno nuevo puede tener como consecuencias una confusa mezcla de ambos

EJEMPLOS del comando NEW:

```
NEW         (Borra el programa y todas las variables)
10 NEW      (Realiza la operación NEW, borrando y deteniendo
            el programa)
```

NEXT

TIPO:Instrucción
FORMATO:NEXT [<contador>] [<contador>]...

Acción: La instrución NEXT se usa conjuntamente con FOR para establecer el fin de un bucle FOR...NEXT. La instrucción NEXT no tiene porque ser la última instrucción física de un bucle, pero es siempre la última instrucción ejecutada en el mismo. El <contador> es el nombre de la variable de control usada con FOR al iniciar el bucle. Un solo NEXT puede servir para varios bucles si detrás se colocan los nombres de las variables de control de cada uno de ellos. Para que esto funcione es preciso colocar en primer lugar la variable del bucle más interior, y de ahí hacia afuera, dejando como última variable la del bucle que encierra a todos los demás. Cuando se usa NEXT de esta forma, cada variable debe separarse mediante una coma. Se permiten hasta nueve bucles "anidados", es decir, uno

dentro de otro. Si se omite el nombre de variable el ordenador asume que es la del bucle más cerrado.
Cuando se ejecuta NEXT, el valor del contador se aumenta en 1 o en el valor opcional de STEP. Entonces se comprueba si se ha alcanzado el límite de cuenta para finalizar el bucle. Un bucle finaliza cuando el valor del contador es superior al límite establecido.

EJEMPLOS de la instrucción NEXT:

```
10 FOR J=1 TO 5:FOR K=10 TO 20:FOR N=5 TO -5 STEP -1
20 NEXT N,J,K                (Fin de los bucles anidados)

10 FOR L=1 TO 100
20 FOR M=1 TO 10
30 NEXT M
40 NEXT L                    (Advierta que los bucles NO
                             pueden cruzarse unos con otros)

10 FOR A=1 TO 10
20 FOR B=1 TO 20
30 NEXT
40 NEXT                      (Advierta que no es necesario
                             colocar nombres de variables)
```

NOT

TIPO:Operador lógico
FORMATO:NOT <expresión>

Acción: El operador lógico NOT "complementa" el valor de cada bit en su unico operando, produciendo un entero "complementario â dos". En otras palabras, NOT dice realmente, "si esto no es...". Cuando se trabaja con números de coma flotante, los operandos son convertidos a enteros y los decimales se pierden. El operador NOT también puede ser usado en una comparación para invertir el valor verdadero/falso que se obtiene como resultado de un test de comparación y por esto invierte el significado de la comparación. En el primer ejemplo, si el complemento de "AA" es igual a "BB" y si "BB" no es igual a "CC", entonces la expresión es cierta.

EJEMPLOS del operador NOT:

```
10 IF NOT AA=BB AND NOT(BB=CC)THEN...

NN%=NOT 96:PRINT NN%
-97
```

NOTA: Para encontrar el valor de NOT utilice la expresión X=(-(X+1)) (El complementario de un número entero es el bit complementario más 1.)

ON

TIPO:Instrucción
FORMATO:ON <variable> GOTO/GOSUB <número de línea>[<número de línea>]...

Acción: La instrucción ON se usa para conducir el programa a uno de varios números de línea, dependiendo del valor de la variable. El valor de la variable puede ir de 0 al numero de lineas dado en la instrucción. Si el valor no es entero, se desprecia la parte decimal. Por ejemplo, si el valor de la variable es 3, ON dirigirá el programa al tercer numero de línea de la instrucción.
Si el valor de la variable es negativo aparece el mensaje de error **ILLEGAL QUANTITY**. Si el número es cero, o mayor que la cantidad de lineas de la lista, el programa "ignora" la instrucción y ejecuta la siguiente.
ON es en realidad una variante de la instrucción IF...THEN. En lugar de usar varios IF...THEN, cada uno enviando a una linea determinada, la instrucción ON lo hace con una sola linea de programa. Viendo el primer ejemplo se dará cuenta de que una sola instrucción ON reemplaza a cuatro instrucciones IF...THEN.

EJEMPLOS de la instrucción ON:

```
ON -(A=7)-2*(A=3)-3*(A<3)-4*(A>7)GOTO 400,900,1000,100

ON X GOTO 100,200,300,320

ON X+3 GOSUB 9000,20,9000

100 ON NUM GOTO 150,300,340,410
500 ON SUM/2+1 GOSUB 50,80,20
```

OPEN

TIPO:Instrucción de E/S
FORMATO:OPEN <núm. fichero>,[<periférico>][,<dirección>] ["<nombre fichero>][,<tipo>][,<modo>"]

Accion: Esta instrucción abre un canal para comunicarse con un periférico. Sin embargo, usted no necesita todos los parámetros del formato en cada instrucción OPEN. Algunas de las instrucciones OPEN requieren solo dos códigos:

1)NUMERO DE FICHERO LOGICO
2)NUMERO DE PERIFERICO

El <núm. fichero> es el número de fichero lógico, al que se refieren las instrucciones OPEN, CLOSE, CMD, GET#, INPUT# y PRINT# para asociarlas con el periférico

utilizado. El número de fichero lógico puede ir de 0 a 255, pudiéndose asignar cualquier número dentro de este rango.

> **NOTA:** Los números de fichero mayores de 127 fueron diseñados realmente para otros usos, por lo que es mejor utilizar números menores de 127 para numoro de fichero

Cada periférico (impresora, cassette, unidad de disco) tiene su propio número de periférico. El número de <periférico> se usa con OPEN para especificar el periférico que se desea utilizar para almacenar o extraer datos. Los periféricos como discos, cassettes e impresoras pueden asimismo tener direcciones secundarias. Piense que estos códigos indican al periférico parte de las operaciones que debe realizar. El número de fichero lógico se usa con cada GET#, INPUT# y PRINT#.

Si el número de <periférico> no se asigna, el ordenador asume que desea leer o grabar información en su unidad de cassette, que posee el número de periferico 1. El nombre de fichero también puede omitirse, pero después no podra buscar el fichero por ningun nombre si en el momento de crearlo no le ha dado uno. Cuando almacena ficheros en el cassette, el ordenador asume que la dirección secundaria es 0, si la omite. (operación de lectura).
La dirección secundaria 1 abre (OPEN) un fichero en cassette para escriturá de datos. La dirección secundaria 2 provoca la escritura de una marca de fin de cinta una vez cerrado el fichero de escritura. El marcador de final de cinta previene de la lectura de datos que no pertenezcan al fichero. En el caso de intentarlo apareceria el mensaje de error **?DEVICE NOT PRESENT**.
Para los ficheros en disco, se pueden utilizar las direcciones secundarias de 2 a 14, ambas incluidas, pero otros números tienen significado especial para el DOS (Sistema Operativo de Disco). Debe usar siempre una dirección secundaria cuando utilice la instrucción OPEN con el disco. (Para más detalles sobre el DOS consulte el manual de su unidad de disco.)
El <nombre de fichero> es una cadena de 1 a 16 caracteres, opcional en el cassette y la impresora. Si el <tipo> de fichero se omite, el ordenador asume que se trata de un fichero de programa, a menos que se encuentre el <modo>. Los ficheros secuenciales se abren para lectura (<modo>=R) a menos que especifique que el fichero debe ser de escritura. (<modo>=W). Un <tipo> de fichero puede ser usado para abrir (OPEN) un fichero.
Si intenta acceder a un fichero sin haberlo abierto previamente mediante OPEN, recibirá el mensaje de error **?FILE NOT OPEN**. Si intenta OPEN para la lectura de un fichero inexistente, el mensaje de error **?FILE NOT FOUND** se mostrará en pantalla. Si se intenta abrir un fichero de escritura en disco, y existe otro con el mismo nombre se mostrara el mensaje de error **?FILE EXISTS**. No se comprueba un error de este tipo al trabajar con cassette, por lo que debe asegurarse de que la cinta está en la posición correcta, para que no pueda borrar accidentalmente datos importantes. Si se abre un fichero y se intenta volverlo a abrir mediante otra instrucción OPEN se producirá el error **FILE OPEN**. (Consulte los manuales de los distintos periféricos para más detalles).

EJEMPLOS de las instrucciones OPEN:

10 OPEN 2,8,4,"SALIDA AL DISCO, SEQ, W"	(Abre fichero secuencial en disco).
10 OPEN 1,1,2,"ESCRITURA CASS."	(Escritura; Marca de fin de cinta).
10 OPEN 50,0	(Entrada por teclado)
10 OPEN 12,3	(Salida por pantalla)
10 OPEN 130,4	(Salida por impresora)
10 OPEN 1,1,0,"NOMBRE"	(Lectura desde cassette).
10 OPEN 1,1,1,"NOMBRE"	(Escritura en cassette)
10 OPEN 1,2,0,CHR$(10)	(Apertura de un canal RS-232)
10 OPEN 1,4,0,"CADENA"	(Mayúsculas/Gráficos a impresora)
10 OPEN 1,4,7,"CADENA"	(Mayúsculas/Minúsculas a impresora)
10 OPEM 1,5,0,"CADENA"	(Mayúsculas/Gráficos a la impresora con un número de periférico 5)
10 OPEN 1,8,15,"COMANDO"	(Envia un comando al disco)

OR

TIPO:Operador lógico
FORMATO:<operando>OR<operando>

Acción: Así como los operadores de comparación se usan para tomar decisiones sobre el curso del programa, los operadores lógicos pueden conectar dos o más comparaciones y devolver un valor verdadero/falso que se puede usar en la toma

de decisiones. Cuando se usa en cálculos, el OR lógico da un bit a 1 si uno o los dos bits correspondientes a los operandos están a 1. Esto puede producir como resultado un número entero, dependiendo de los operandos. Cuando se usa en comparaciones, el operador lógico OR se usa para unir dos expresiones en una expresión compuesta. Si una de las dos expresiones es verdadera, el valor de la expresión combinada es verdadero (-1). En el primer ejemplo, si AA es igual a BB o (OR) si XX es igual a 20, la expresión es cierta.
Los operadores lógicos trabajan convirtiendo a los operandos en un número de 16 bits, con signo, en el rango comprendido entre -32768 y +32767. Si los operandos no están en este rango se produce un error. Cada bit del resultado es determinado por los correspondientes bits de los operandos.

EJEMPLOS del operador OR:

100 IF (AA=BB)OR(XX=20)THEN....	
230 KK%=64 OR 32:PRINT KK%	(El valor binario de 64 es 01000000 y el de 32 es 00100000)
	(El ordenador responde con el valor binario 01100000 01100000=96.)

PEEK

TIPO:Función entera
FORMATO:PEEK(<número>)

Acción: Devuelve un entero entre 0 y 255, producto de la lectura de una posición de memoria, que debe estar comprendida entre los números 0 y 65535. Si se intenta otro número aparecerá el mensaje de error **?ILLEGAL QUANTITY**.

EJEMPLOS de la función PEEK:

10 PRINT PEEK (53280) AMD15	(Da como resultado el código de color del borde de la pantalla.)
5 A%=PEEK(45)+PEEK(46)*256	(Resultado: posición de inicio de la zona de variables BASIC.)

POKE

TIPO:Instrucción
FORMATO:POKE<posición>,<valor>

Acción: La instrucción POKE se usa para colocar un valor de 8 bits en una posición de memoria o un registro de entrada/salida. La <posición> es una expresión aritmética que dé como resultado un número entre 0 y 65535. El <valor> es una expresión o número comprendido entre 0 y 255. Si alguno de ambos valores está fuera de rango, se producirá el error **?ILLEGAL QUANTITY**.
Las instrucciones POKE y PEEK son útiles para almacenar datos, controlar displays gráficos o sonido, cargar rutinas en lenguaje máquina y pasar argumentos y resultados desde y hacia subrutinas en lenguaje máquina. Además, varios parámetros del Sistema Operativo pueden ser examinados mediante PEEK y modificados mediante POKE. En el Apéndice G se encuentra un mapa de posiciones de memoria útiles, y las páginas 256-272 contienen el mapa de memoria completo de su Commodore 64.

EJEMPLOS de la instrucción POKE:

POKE 1024,1	(Coloca una "A" en la posición de pantalla número 1.)
POKE 2040,PTR	(Actualiza el puntero de Sprite #0.)
10 POKE RED,32	
20 POKE 36879,8	
2050 POKE A,B	

POS

TIPO:Función Entera
FORMATO:POS(<número>)

Acción: Le dice la posición actual del cursor dentro de la línea lógica de 40 caracteres. Puesto que el Commodore 64 dispone de una pantalla de 80 caracteres, cualquier posición entre 40 y 79 se refiere a la segunda línea de la pantalla. El argumento numérico es ignorado.

EJEMPLOS de la función POS:

```
1000 IF POS(0)>38 THEN PRINT CHR$(13)
```

PRINT

TIPO:Instrucción
FORMATO:PRINT[<variable>][<,/;><variable>]...

Acción: La instrucción PRINT se usa normalmente para imprimir datos en pantalla. Sin embargo, la instrucción CMD puede desviar estos datos a otros

periféricos. Las variables que siguen a PRINT pueden ser expresiones de cualquier tipo. Si no se encuentra una lista de variables, se imprime una línea en blanco. La posición de cada dato impreso se determina por los signos de puntuación usados para separar los mismos.
Los caracteres de puntuación que puede usar son: espacios, comas y punto y coma (;). La línea lógica de 80 caracteres está dividida en 8 zonas de 10 espacios cada una. En la lista de expresiones, una coma entre datos causa que éstos se impriman al inicio de la siguiente zona. Un punto y coma causa que el próximo valor se imprima justo detrás del anterior. Sin embargo, hay dos excepciones a la siguiente regla:
1)Los datos numéricos se imprimen con un espacio después del número.
2)Los números positivos se imprimen precedidos también de un espacio.
Cuando no coloque un signo de puntuación entre dos cadenas, el ordenador asume que hay un punto y coma. Sin embargo, la presencia de espacios entre una cadena y un dato numérico o entre dos datos numéricos causa la interrupción de la impresión sin que se imprima el segundo dato.
Si al final de la lista de salida se encuentra una coma o punto y coma, la próxima instrucción PRINT se ejecutará en la misma línea, siguiendo las reglas anteriores. Si no se encuentra ningún signo al final de un print, se imprimirá un retorno del carro, lo que causa que la próxima instrucción PRINT se ejecute al principio de la próxima línea. Si la salida se dirige a la pantalla y los datos ocupan más de 40 caracteres, la impresión continuará al principio de la próxima línea.
No hay ninguna otra instrucción en BASIC tan versátil y variada como PRINT. Hay muchos símbolos, funciones y parámetros asociados a esta instrucción, que puede ser considerada como un lenguaje propio dentro del BASIC. Un lenguaje diseñado especialmente para escribir en la pantalla.

EJEMPLOS de la instrucción PRINT:

1)

```
5 X=5
10 PRINT -5*X,X-5,X+5,X-5
-25          0          10          3125
```

2)

```
5 X=9
10 PRINT X;"AL CUADRADO ES";X*X;"Y";
20 PRINT X;"AL CUBO ES"X^3

9 AL CUADRADO ES 81 Y 9 AL CUBO ES 729
```

3)

```
90 AA$="ALPHA":BB$="BAKER":CC$="CHARLIE":DD$="DOG":
   EE$="ECHO"
100 PRINT AA$;BB$;CC$;DD$,EE$

ALPHABAKERCHARLIEDOG       ECHO
```


Modo Comillas

Cuando se escriben comillas (SHIFT 2), se detienen las operaciones de control del cursor y se inicia la impresión de caracteres en inverso que señalan los distintos movimientos del mismo. Esto le permite programar estos controles de cursor, de forma que el texto a imprimir se muestre en la posición deseada. La tecla INST/DEL es el único control del cursor no afectado por el "modo comillas".

1. Movimiento del Cursor.

Los controles del cursor que pueden ser "programados" en modo comillas son:

TECLA	APARECE COMO
CLR/HOME	S
SHIFT CLR/HOME	♥
⇑ CRSR ⇓	Q
SHIFT ⇑ CRSR ⇓	○
⇐CRSR⇒	]
SHIFT ⇐CRSR⇒	‖

Si desea imprimir la palabra HELLO diagonalmente desde la esquina superior izquierda de la pantalla, debe escribir:

PRINT "CLR/HOME H CRSR E CRSR L CRSR L CRSR O"

Lo que aparece como:

PRINT "S H Q E Q L Q L Q O"

2.Caracteres Inversos

Apretando la tecla CTRL y la tecla 9, aparece el caracter R entre comillas. Esto produce que todos los caracteres siguientes se impriman en VIDEO INVERSO (como un negativo fotográfico). Para finalizar este modo de impresión pulse CTRL 0, o imprima un RETURN (CHR$(13)). (Es suficiente terminar la instrucción PRINT sin coma o punto y coma.)

3.Controles de Color

Pulsando la tecla CTRL o la C= con alguna de las 8 teclas de color se forma un carácter especial inverso entre las comillas. Cuando se ejecuta el PRINT, los datos aparecerán en el color al que se refiere el carácter.

TECLA	COLOR	APARECE COMO
CTRL 1	NEGRO	
CTRL 2	BLANCO	
CTRL 3	ROJO	
CTRL 4	CYAN	
CTRL 5	PURPURA	
CTRL 6	VERDE	
CTRL 7	AZUL	
CTRL 8	AMARILLO	
C= 1	NARANJA	
C= 2	MARRON	
C= 3	ROJO CLARO	
C= 4	GRIS 1	
C= 5	GRIS 2	
C= 6	VERDE CLARO	
C= 7	AZUL CLARO	
C= 8	GRIS 3	

Si desea imprimir la palabra HELLO en cyan y THERE en blanco, escriba:

PRINT "CTRL 4 HELLO CTRL 2 THERE"

lo que aparece como:

PRINT " HELLO E THERE"

4. Modo de Inserción

Los espacios creados mediante la tecla INST/DEL tienen las mismas características que en modo comillas. La única diferencia es que DEL, que tenía una función normal incluso en modo comillas, ahora crea una T. E INST, que creaba un símbolo especial en modo comillas, ahora inserta espacios normalmente. A causa de esto, es posible crear una instrucción PRINT conteniendo la función de borrado (DEL). He aquí un ejemplo de esto:

10 PRINT"HELLO" INST/DEL SHIFT INST/DEL SHIFT INST/DEL INST/DEL INST/DEL P"

que aparece como:

10 PRINT"HELLO T T P"

Cuando se ejecute la línea anterior, se mostrará la palabra HELP, puesto que se han borrado los dos últimos caracteres de la palabra HELLO y se ha añadido una P en su lugar.

> **ATENCION:** Los DELetes funcionan en el listado igual que si se imprimieran, por lo que editar una línea con dichos caracteres puede ser bastante difícil.

La condición "modo de inserción" se termina al pulsar RETURN o SHIFT RETURN, o cuando se han cubierto todos los espacios insertados.

5. Otros Caracteres Especiales.

Hay otros caracteres con funciones especiales, aunque no son accesibles fácilmente mediante el teclado. Para colocarlos entre comillas, debe dejar espacios vacíos en la línea, pulsar RETURN o SHIFT RETURN, volver a estos espacios, pulsar CTRL RVS/ON, para iniciar la impresión inversa, y escribir las teclas mostradas abajo:

Función	Escriba		
SHIFT/RETURN	SHIFT	M	
	N		
Cambiar a minúsculas Cambiar a mayúsculas	SHIFT	N	
Desactivar teclas de cambio de Mayúsc./Minúsc.	H		
Activar teclas de cambio de Mayúsc./Minusc.	I		

El SHIFT RETURN trabaja en el listado igual que en la impresión, por lo que la edición es imposible si se usa este carácter. Además, el listado aparecerá de forma muy extraña.

PRINT#

TIPO:Instrucción de E/S
FORMATO:PRINT#<número de fichero>[<variable>][<,/;><variable>]...

Acción: La instrucción PRINT# se usa para escribir datos en un fichero lógico. Debe usar el mismo número de la instrucción OPEN que se utilizó para el fichero. La salida se dirige al número de periférico usado en la instrucción OPEN. Las expresiones de la lista pueden ser de cualquier tipo. Los caracteres de puntuación

entre datos son los mismos que para la instrucción PRINT y se usan de la misma forma. El efecto de la puntuación es distinto en dos aspectos fundamentales Cuando se usa en ficheros sobre cassette, la coma, en lugar de espaciar imprimiendo por zonas, tiene el mismo efecto que punto y coma. Sin embargo, cuando no se usan signos de puntuación entre datos, el efecto de espaciado es el mismo. Los datos son escritos como una sola cadena de caracteres Los datos numericos estan seguidos de un espacio, y si son positivos estará precedidos de otro.

Si no hay puntuación al final de la lista, se envía un retorno de carro. Si la lista se termina con coma o punto y coma, el retorno de carro es suprimido. Independientemente de la puntuación, el próximo dato empieza en el siguiente espacio disponible. La alimentación de línea actúa como stop durante la ejecución de INPUT#, dando como resultado una cadena vacía. La alimentación de línea puede ser eliminada o compensada como se verá luego.

La forma más fácil de escribir más de una variable en un fichero sobre disco o cinta es asignar a una variable el código CHR$(13) y usarla como separador entre datos.

EJEMPLOS de la instrucción PRINT#

1)

```
10  OPEN 1,1,1,"FICHERO CINTA"
20 R$ = CHR$(13)
30 PRINT# 1,1;R$;2;R$;3;R$;4;R$;5
40 PRINT# 1,6
50 PRINT# 1,7
```

(Cambiando CHR$(13) por CHR$(44) colocará una coma entre variables. CHR$(59) coloca un ";" entre ellas.)

2)

```
10 CO$=CHR$(44): CR$=CHR$(13)
20 PRINT#1, "AAA"CO$"BBB",
   "CCC";"DDD";"EEE"CR$
   "FFF"CR$;
30 INPUT#1, A$,BCDE$,F$
```

AAA,BBB CCCDDDEEE
(Retorno de carro)
FFF(Retorno de carro)

3)

```
 5 CR$=CHR$(13)
10 PRINT#2, "AAA";CR$;"BBB"
20 PRINT#2, "CCC";

30 INPUT#2, A$,B$,DUMMY$,C$
```

(10 espacios)AAA
BBB
(10 espacios)CCC

READ

TIPO:Instrucción
FORMATO:READ <variable> [<variable>]...

Acción: La instrucción READ se usa para asignar a variables las constantes contenidas en las instrucciones DATA. Los datos contenidos en DATA deben concordar con el tipo de variable de READ, si esto no ocurre se producirá el error BASIC **?SYNTAX ERROR***. Las variables de la lista DATA deben estar separadas por comas.

Una sola instrucción READ puede acceder a una o más instrucciones DATA, a las que se accede por orden (Vea DATA), o una sola instrucción DATA puede ser leida varias veces por READ. Si se ejecutan más instrucciones READ que el número de elementos en las instrucciones DATA, se imprime el mensaje de error **?OUT OF DATA**. Si el número de variables especificado es menor que el número de elementos DATA, los subsiguientes READ continuarán a partir del punto en que finalizó la última lectura. (Vea RESTORE).

*NOTA: El ?SYNTAX ERROR aparece con el número de línea de la instrucción DATA, no de la READ.

EJEMPLOS de la instrucción READ:

```
110  READ A,B,C$
120  DATA 1,2,HELLO

100  FOR X=1 TO 10:READ A(X):NEXT
200  DATA 3.08,5.19,3.12,3.98,4.24
210  DATA 5.08,5.55,4.00,3.16,3.37

1  READ CIUDAD$,ESTADO$,ZIP
5  DATA DENVER,COLORADO,80211
```

REM

TIPO:Instrucción
FORMATO:REM [<comentario>]

Acción: La instrucción REM permite una mayor comprensión del programa cuando se lista. Sirve para recordarle a usted cómo y con qué ideas realizó cada parte del programa. Por ejemplo, puede usarla para recordar el uso de cada variable, u otra información útil. El contenido de una instrucción REM puede ser de cualquier tipo: palabras, números, signos de puntuación (incluido ":") e incluso palabras BASIC.

La instrucción REM y todo lo que le sigue en el mismo número de línea es ignorado por el intérprete BASIC, pero al listar un programa se muestra tal como se imprimió. Un GOTO o GOSUB puede referirse a una línea REM, y el programa continuará en la próxima línea más alta que contenga instrucciones ejecutables.

EJEMPLOS de la instrucción REM:

```
10  REM CALCULA MEDIA DE VELOCIDAD
20  FOR X=1 TO 20:REM BUCLE PARA 20 VALORES
30  SUM=SUM+VEL(X):NEXT
40  MDV=SUM/20
```

RESTORE

TIPO:Instrucción
FORMATO:Restore

Acción: El BASIC mantiene un puntero interno que indica la próxima constante DATA a leer. Este puntero puede ser inicializado a la primera constante DATA mediante RESTORE. RESTORE puede ser usado en cualquier parte del programa para iniciar la lectura de constantes DATA desde el principio.

EJEMPLOS de la instrucción RESTORE:

```
100  FOR X=1 TO 10:READ A(X):NEXT
200  RESTORE
300  FOR Y=1 TO 10:READ B(Y):NEXT
400  DATA 33,6,256,44,35
500  DATA 2,65,221,78,21
```

(Construye dos tablas con datos idénticos)

```
10  DATA 1,2,3,4
20  DATA 5,6,7,8
30  FOR L=1 TO 8
40  READ A:PRINT A
50  NEXT:RESTORE
60  FOR L=1 TO 8
70  READ A:PRINT A
80  NEXT
```


RETURN

TIPO:Instrucción
FORMATO:RETURN

Acción: La instrucción RETURN se usa para terminar una subrutina llamada mediante la instrucción GOSUB. Si el programa contiene subrutinas anidadas (unas dentro de otras), cada GOSUB deberá estar emparejado con al menos un RETURN. Una subrutina puede contener cualquier número de RETURNS, pero el primero que se encuentre producirá la salida de la misma.

EJEMPLO de la instrucción RETURN:

```
10  PRINT "ESTO ES EL PROGRAMA"
20  GOSUB 1000
30  PRINT "EL PROGRAMA CONTINUA"
40  GOSUB 1000
50  PRINT "MAS PROGRAMA"
60  END
1000  PRINT"ESTO ES LA SUBRUTINA":RETURN
```

RIGHT$

TIPO:Función de Cadena
FORMATO:RIGHT$(<cadena>,<número>)

Acción: La función RIGHT$ da como resultado una subcadena tomada de la parte derecha del argumento «cadena». La longitud de la subcadena está especificada por el número del argumento, que debe estar comprendido entre 0 y 255. Si el valor de la expresión numérica es cero, se devuelve una cadena vacia (""). Si el valor es mayor que la longitud total de la cadena, el resultado es la cadena entera.

EJEMPLOS de la función RIGHT$:

```
10  MSG$="COMMODORE COMPUTERS"
20  PRINT RIGHT$(MSG$,9)
RUN

COMPUTERS
```

RND

TIPO:Función de coma flotante
FORMATO:RND(<número>)

Acción: RND crea un número aleatorio en coma flotante entre 0.0 y 1.0. El ordenador genera una secuencia de números aleatorios realizando cálculos sobre un número de inicio, que en la jerga de los programadores se llama "semilla". Durante las operaciones que siguen al encendido del ordenador se fija la "semilla"

para RND. El argumento númerico no tiene valor para el cálculo, excepto el signo (positivo, cero o negativo).
Si el argumento es positivo, se produce una secuencia de números "pseudoaleatorios", empezando por el valor de la "semilla". Se producirán distintas secuencias con distintas semillas, pero cada secuencia es repetible empezando por la misma semilla. Tener una secuencia conocida de números "aleatorios" puede ser útil para probar programas.
Si escoge el argumento cero (0), RND genera el número directamente a partir del sistema de reloj del ordenador. Los argumentos negativos causan que se utilice siempre la misma semilla.

EJEMPLOS de la función RND:

220 PRINT INT(RND(0)*50)	(Números enteros entre 0 y 49)
100 X=INT(RND(1)*6)+INT(RND(1)*6)+2	(Simula dos dados)
100 X=INT(RND(1)*1000)+1	(Enteros entre 1 y 1000)
100 X=INT(RND(1)*150)+100	(Enteros entre 100 y 249)
100 X=RND(1)*(U−L)+L	(Números aleatorios entre los límites U (superior) y L (inferior))

RUN

TIPO:Comando
FORMATO:RUN[<número de línea>]

Acción: El comando RUN se usa para iniciar la ejecución de un programa en memoria. El comando RUN implica la ejecución de una operación CLR antes de ejecutar el programa. Puede evitar esta operación usando GOTO o CONT en lugar de RUN. Si se especifica <número de línea>, el programa empezará por la misma. Si no se especifica, el programa se inicia por la línea con el número más bajo. Si el <número de línea> no existe se devuelve el error **UNDEF'D STATEMENT**.
Un programa se para y el BASIC vuelve al modo directo cuando se encuentra una instrucción STOP o END, cuando se ejecuta la última línea del programa o cuando se produce un error en la ejecución.

EJEMPLOS del comando RUN:

RUN	(Empieza en la primera línea del programa)
RUN 500	(Empieza en la línea 500)
RUN X	(Empieza en la línea X, o **UNDEF'D STATEMENT ERROR** si la línea X no existe)

SAVE

TIPO:Comando
FORMATO:SAVE["<nombre fichero>"][,<periférico>]

[<dirección>]

Acción: El comando SAVE se usa para guardar como fichero en disco o cinta un programa almacenado actualmente en la memoria del ordenador. SAVE sólo afecta al programa mientras se ejecuta. Una vez ejecutado el comando, el programa seguirá en memoria exactamente igual a como estaba antes. El fichero creado será "PRG" (programa). Si se omite el <periférico> (número del periférico), el ordenador asume que el programa se desea grabar en cassette, que tiene el número de periférico 1. Si el número de periférico es 8, el programa se grabará en la unidad de disco. El comando SAVE se puede ejecutar dentro de un programa, continuando la ejecución de la próxima instrucción una vez completado SAVE.

Los programas en cinta se almacenan dos veces consecutivas, de forma que el Commodore 64 pueda chequear errores cuando cargue el programa (LOAD) Cuando guarda (SAVE) programas en cinta, el <nombre de fichero> y la <dirección> secundaria son opcionales. Sin embargo, asignar un nombre entrecomillado (o una variable) al programa permite que el Commodore 64 encuentre con más facilidad el programa a la hora de ser cargado. Si se omite el nombre de programa, cuando se quiera cargar NO se podrá especificar ningún nombre.
La dirección secundaria 1 indica al KERNAL que cuando cargue posteriormente el programa lo haga en las mismas posiciones de memoria que ocupaba cuando fue grabado, en lugar de a partir de la 2048, si es necesario. La dirección secundaria 2 coloca al final del programa un indicador de fin de cinta. La dirección secundaria 3 combina ambas funciones.
Cuando se graba un programa en disco, el <nombre de fichero> se debe incluir siempre.

EJEMPLOS del comando SAVE:

SAVE	(Graba en cinta sin nombre)
SAVE "ALPHA", 1	(Graba en cinta con el nombre "alpha")
SAVE "ALPHA", 1, 2	(Graba "alpha" con marca de fin de cinta)
SAVE "FUN.DISK",8	(Graba en disco (el número 8 es el número de periférico del disco))
SAVE A$	(Guarda en cinta con el nombre contenido en A$)
10 SAVE "HI"	(Graba en cinta y prosigue la ejecución en la siguiente línea)
SAVE "ME",1,3	(Graba en cinta un programa no relocatable con marca de fin de cinta)

SGN

TIPO:Función entera
FORMATO:SGN(<número>)

Acción: SGN le da un valor entero dependiendo del signo del argumento. Si el argumento es positivo, el resultado es 1. Si el argumento es 0, el resultado es también 0. Si es negativo, el resultado es −1.

EJEMPLO de la función SGN:

```
90 ON SGN(DV)+2 GOTO 100,200,300
```
(Salta a 100 si DV es negativo, a 200 si DV=0 y a 300 si DV es positivo)

SIN

TIPO:Función de coma flotante
FORMATO:SIN(<número>)

Acción: SIN le da el seno del argumento, en radianes. El valor de COS(X) es igual a SIN(X+3.14159265/2).

EJEMPLO de la función SIN:

```
235 AA=SIN(1.5):PRINT AA

.997494987
```

SPC

TIPO:Función de cadena
FORMATO:SPC(<número>)

Acción: La función SPC se usa para formatear la salida de datos, ya sea en la pantalla o en un fichero lógico. El argumento de SPC equivale al número de eSPaCios que se imprimen, a partir de la próxima posición libre. Para la pantalla y ficheros en cinta el máximo número de SPC es 255, para ficheros en disco es 254. Para salida a impresora, se imprimirán espacios hasta llegar al número especificado o a la última posición de una línea. LLegado a este punto se imprime un retorno de carro, no imprimiéndose más espacios en la siguiente línea.

EJEMPLO de la función SPC:

```
10 PRINT"RIGHT ";"HERE &";
20 PRINT SPC(5)"OVER"SPC(14)"THERE"

RUN

RIGHT HERE &     OVER              THERE
```


SQR

TIPO:Función de coma flotante
FORMATO:SQR(<número>)

Acción: SQR le da el valor de la Raíz Cuadrada del argumento numérico. El valor del argumento no puede ser negativo, o se producirá el error **?ILLEGAL QUANTITY.**

EJEMPLO de la función SQR:

```
FOR J=2 TO 5:PRINT J*5,SQR(J*5):NEXT

10   3.16227766
15   3.87298335
20   4.47213595
25   5

READY
```

STATUS

TIPO:Función entera
FORMATO:STATUS

Acción: Devuelve el estado de la última operación de E/S efectuada en un fichero abierto previamente. El STATUS puede ser leído desde cualquier periférico. La palabra reservada STATUS (o simplemente ST), es una variable definida por el sistema en la que el KERNAL coloca el estado de las operaciones de E/S. A continuación se muestra una tabla conteniendo los valores que puede contener ST y su significado:

Pos. bit ST	Valor numérico de ST	Lectura Cassette	Bus serie L/E	Verify y Load C.
0	1		Escritura fuera de tiempo.	
1	2		Lectura fuera de tiempo.	
2	4	Bloque corto.		Bloque cor.
3	8	Bloque largo.		Bloque lar.
4	16	Error de lectura irrecuperable.		Cualquier desacuerdo.
5	32	Error de revisión.		Error de revisión.
6	64	Fin de fichero	EOI	
7	−128	Fin de cinta	Periférico no presente	Fin de cinta

EJEMPLOS de la función STATUS (ST)

```
10  OPEN 1,4:OPEN 2,8,4,"MASTER FILE,SEQ,W"
20  GOSUB 100:REM CONPRUEBA ESTADO
30  INPUT#2,A$,B,C
40  IF STATUS AND 64 THEN 80:REM FIN DE FICHERO
50  GOSUB 100:REM COMPRUEBA ESTADO
60  PRINT#1,A$,B;C
70  GOTO  20
80  CLOSE1:CLOSE2
90  GOSUB 100:END
100 IF ST>0 THEN 9000:REM COMPRUEBA ERRORES DE E/S
110 RETURN
```

STEP

TIPO:Instrucción
FORMATO:[STEP<expresión>]

Acción: La palabra clave opcional STEP sigue al <valor de final> en la instrucción FOR. Esta instrucción define el valor del incremento en la variable de control. Como incremento se puede usar cualquier valor excepto 0. Si se omite STEP, el incremento se realizará de 1 en 1. Cuando se encuentra la instrucción NEXT, se

incrementa la variable de control del bucle en la cantidad indicada por la expresión que sigue a STEP. Entonces se comprueba el valor de dicha variable con el valor final del bucle para terminar –si fuera el caso– la ejecución del mismo. (Vea la instrucción FOR para más detalles.)

NOTA: El valor de STEP no puede ser cambiado una vez iniciado el bucle.

EJEMPLOS de la instrucción STEP:

```
25  FOR  XX=2  TO  20  STEP  2    (El bucle se repite 10 veces)
35  FOR  A=0   TO  -20  STEP  -2  (El bucle se repite 11 veces)
```

STOP

TIPO:Instrucción
FORMATO:STOP

Acción: La instrucción STOP se usa para detener la ejecución de un programa y volver al modo directo. La pulsación de la tecla RUN/STOP produce el mismo efecto. Se imprime el mensaje **?BREAK IN LINE nnnnn**, seguido de **READY**. "nnnnn" es el número de línea en que se interrumpió el programa. Todos los ficheros abiertos siguen abiertos y las variables permanecen intactas para que puedan ser examinadas. El programa puede ser continuado mediante CONT o GOTO nnnnn.

EJEMPLOS de la instrucción STOP:

```
10 INPUT#1,AA,BB,CC
20 IF AA=BB AND BB=CC THEN STOP
30 STOP
```

(Si las tres variables son iguales aparecerá:)

```
BREAK IN LINE 20
BREAK IN LINE 30      (Si no son iguales)
```

STR$

TIPO:Función de Cadena
FORMATO:STR$(<numérico>)

Acción: STR$ convierte en cadena el valor numérico de la expresión. Los números estarán seguidos de un espacio, y –si son positivos– precedidos por otro.

EJEMPLO de la función STR$

```
100 FLT=1.5E4:ALPHA$=STR$(FLT)
110 PRINT FLT,ALPHA$

15000          15000
```

SYS

TIPO:Instrucción
FORMATO:SYS<posición de memoria>

Acción: Esta es la forma más corriente de mezclar un programa en BASIC con una rutina en lenguaje maquina. El programa en lenguaje máquina se ejecuta a partir de la posición de memoria especificada en el SYS. La Instrucción SYS se puede ejecutar en modo directo o dentro de un programa para que el microprocesador tome el control de un programa en lenguaje máquina existente en la memoria. La posición de memoria puede ser el resultado de una expresión, y debe estar comprendida entre 0 y 65535, con lo que el programa puede estar en RAM o en ROM.
Cuando use la instrucción SYS debe terminar su rutina en lenguaje máquina con la instrucción RTS (ReTorno de Subrutina), de forma que una vez terminado el programa en C/M, el ordenador vuelva al BASIC y ejecute la siguiente instrucción.

EJEMPLOS de la instrucción SYS:

SYS 64738	(Ejecuta un programa en ROM que reinicializa el sistema).
10 POKE 4400,96:SYS 4400	(Ejecuta la instrucción en lenguaje máquina de la posición 4400. Puesto que el código 96 equivale a la instrucción en C/M RTS, se vuelve inmediatamente al BASIC.

TAB

TIPO:Función de cadena
FORMATO:TAB(<número>)

Acción: La función TAB mueve el cursor a la posición relativa en pantalla indicada por el argumento, a partir de la primera posición de la izquierda de la línea. El valor del argumento debe estar comprendido entre 0 y 255. La función TAB debe ser usada con la instrucción PRINT, ya que no funciona con la instrucción PRINT# a un fichero lógico.

EJEMPLO de la función TAB:

```
100 PRINT"NOMBRE" TAB(25)"IMPORTE".PRINT
110 INPUT#1,NOM$,IMP$
120 PRINT NAM$ TAB(25)IMP$

NOMBRE IMPORTE

G.T JONES 25.
```


TAN

TIPO:Función de coma flotante
FORMATO:TAN(<número>)

Acción: Devuelve la tangente del valor de la expresión <numérica> en radianes. Si la función TAN queda fuera de las posibilidades de calculo del ordenador, es mostrado el mensaje **?DIVISION BY ZERO.**

EJEMPLO de la función TAN:

```
10 XX=0.785398163:YY=TAN(XX):PRINT YY
```

TIME

TIPO:Función numérica
FORMATO:TI

Acción: La función TI lee el reloj interno del microprocesador. El reloj es puesto a cero (inicializado) cuando usted conecta el ordenador. El contador aumenta una unidad cada 1/60 de segundo. Este reloj a intervalos de 1/60 de segundo se detiene durante las operaciones de E/S con el lector de cinta.

EJEMPLO de la función TI:

```
10 PRINT TI/60 "SEGUNDOS DESDE LA CONEXION
   DEL ORDENADOR"
```

TIME$

TIPO:Función de cadena
FORMATO:TI$

Acción: El reloj TI$ aparece y trabaja como un RELOJ REAL mientras el ordenador está conectado. Para actualizar TI$ se utiliza el reloj interno del hardware. TI$ se presenta como una cadena de 6 caracteres, que indican horas, minutos y segundos. Se puede asignar un punto de arranque arbitrario para ajustar el reloj a la hora real (de forma similar a como ajusta su reloj de pulsera). El valor de TI$ no es exacto después de una operación de E/S con el cassette.

EJEMPLO de la función TI$

```
1 TI$="000000":FOR J=1 TO 10000:NEXT:PRINT TI$

000011
```

USR

TIPO:Función de coma flotante
FORMATO:USR(<número>)

Acción: La función USR realiza el salto a una subrutina en lenguaje máquina en la posición de inicio definida por el puntero de las posiciones 785-786. Antes de utilizar esta función se deben colocar los valores adecuados a la dirección de inicio de la subrutina en las posiciones 785-786, mediante POKE. A menos que se haya realizado esta operación, se recibirá el mensaje de error ?ILLEGAL QUANTITY. El valor del argumento <numérico> se almacena en el acumulador de coma flotante que se inicia en la posición 97, para poder acceder a él desde código máquina, y el resultado de la función USR es el valor que queda una vez ejecutada la subrutina, volviendo al BASIC.

EJEMPLOS de la función USR:

```
10 B=T*SIN(Y)
20 C=USR(B/2)
30 D=USR(B/3)
```

VAL

TIPO:Función numérica
FORMATO:VAL(<cadena>)

Acción: Devuelve un VALor numérico representando los datos contenidos en el argumento <cadena>. Si el primer caracter de la cadena que no sea espacio no es el signo más (+), menos (–) o un dígito, el resultado de VAL es cero (0). La conversión de la cadena finaliza cuando ésta se termina o se encuentra un carácter no numérico (excepto el punto decimal o el exponencial e).

EJEMPLO de la función VAL:

```
10  INPUT#1,NAM$,ZIP$
20  IF VAL(ZIP$)<19400 OR VAL(ZIP$)>96699
    THEN PRINT NAM$ TAB(25)"GREATER PHILADEPHIA"
```

VERIFY

TIPO:Comando
FORMATO:VERIFY["<nombre de fichero>"][,<periférico>]

Acción: El comando VERIFY se usa, en modo directo o en un programa, para comparar el programa actualmente en memoria con un fichero de programa en disco o cassette. El uso más adecuado de VERIFY es después de una operación SAVE, para asegurarse de que el programa acabado de grabar se ha almacenado correctamente.
Si se omite el número de <periférico>, el ordenador asume que se utiliza el Datassette (TM), que tiene el número de periférico 1. En los ficheros en cassette, si se omite el <nombre de fichero>, se verifica el primer programa que se encuentre en la cinta. Para los ficheros en disco es necesario indicar el nombre de fichero. Si se encuentran diferencias entre el programa en memoria y el grabado, se mostrará el mensaje ?VERIFY ERROR.
El nombre de programa debe ir entre comillas o bien estar contenido en una variable de cadena. VERIFY se usa también para posicionar la cinta justo detrás de un programa grabado, con lo que se evita el siempre problemático accidente de grabar un programa encima de otro.

EJEMPLOS del comando VERIFY:

VERIFY PRESS PLAY ON TAPE OK SEARCHING FOUND <FILENAME> VERIFYING	(Comprueba el primer programa en cinta)
9000 SAVE "ME",8: 9010 VERIFY "ME",8	(Verifica la correcta grabación en el disco (periférico número 8))

WAIT

TIPO:Instrucción
FORMATO:WAIT<posición>,<máscara-1>[<máscara-2>]

Acción: La instrucción WAIT causa la detención del programa hasta que se encuentre el valor adecuado en la posición de memoria especificada. En otras palabras, WAIT se usa para detener el programa hasta que ocurra una condición determinada, como, por ejemplo, se pulse una tecla. Esto se logra monitorizando el estado de los bits en los registros de E/S. Los parámetros de WAIT pueden ser expresiones numéricas, pero se transformarán a números enteros.
La mayoría de programadores no usan nunca esta instrucción. Se usa principalmente en operaciones de E/S, y para casi nada más.
La instrucción WAIT toma el valor de la posición de memoria y realiza un AND lógico con el valor de la máscara-1. Si se incluye una segunda máscara en la instrucción, se realiza un OR exclusivo entre el resultado de la primera operación y la máscara-2. En otras palabras, la máscara-1 "filtra" los bits que desea chequear. Cuando un bit esta a cero en la máscara-1, el correspondiente bit del resultado siempre sera cero. Mediante la máscara 2 puede consultar una condición "encendido/apagado". Los bits que se deseen chequear como 0 deberán tener un 1 en la posición correspondiente de la máscara–2.
Si los correspondientes bits comparados difieren, el OR exclusivo da un resultado de 1. Si los dos bits tienen el mismo valor, el resultado es 0. Mediante WAIT se

puede entrar en una detención infinita del programa, que se puede anular mediante la pulsación simultanea de RUN/STOP y RESTORE. Pulse primero RUN/STOP y -manteniendola apretada- pulse RESTORE. En el primer ejemplo, el programa se detiene hasta que se pulsa una tecla en el cassette. El segundo ejemplo espera hasta que un Sprite choca con la pantalla.

EJEMPLOS de la instrucción WAIT:

```
WAIT 1, 32, 32
WAIT 53273, 6, 6
WAIT 36868, 144, 16
```

(144 y 16 son máscaras. 144=10010000 en binario y 16 00010000. La instrucción WAIT detiene el programa hasta que el bit 7 esté a 1 o el bit 4 este a 0).

EL TECLADO DEL COMMODORE 64 Y SUS CARACTERISTICAS

El sistema operativo posee un "buffer" de teclado de 10 caracteres que se usa para almacenar las teclas que se pulsan hasta que son procesadas. Este buffer, o "cola" almacena en orden el código de las teclas, de forma que la primera en pulsarse será la primera en ser procesada. Por ejemplo, si mientras se procesa una tecla se pulsa otra, esta segunda tecla (su correspondiente carácter) es almacenada en el buffer, mientras continua el procesado del primer carácter. Una vez el programa ha procesado el primer caracter, se buscan más datos en el buffer de teclado, procesando el siguiente. Si el Sistema Operativo no dispusiera de este buffer, una persona que tecleara a gran velocidad podría perder algunos caracteres.
En otras palabras, el buffer de teclado le permite "escribir por delante", lo que significa que puede responder anticipadamente a INPUT o GET. Cuando pulsa teclas, se almacenan por orden de pulsación en el buffer, esperando ser procesadas por el programa. Esta característica puede ocasionar problemas cuando una pulsacion accidental causa que el programa tome un caracter equivocado del buffer.
Normalmente, las pulsaciones incorrectas no ocasionan problemas puesto que pueden ser corregidas mediante la tecla de CuRSoR a la izquierda CRSR o DELete INST/DEL, y después escribir las letras correctas. Pero si además de pulsaciones incorrectas ha pulsado la tecla RETURN, no será posible efectuar correcciones, ya que si el buffer contiene el código de retorno del carro, los caracteres presentes en él son procesados antes de poder efectuar correcciones. Esta situación se puede evitar usando un bucle para vaciar el buffer de teclado antes de pedir una respuesta del usuario:

```
10 GET JUNK$: IF JUNK$<>""THEN10:REM VACIA EL BUFFER DE
TECLADO (*)
```

(*) POKE 198,0 produce el mismo efecto, siendo más rápido y gastando menos memoria. (N. del T.)

Además de con GET e INPUT, el teclado puede ser leído mediante el contenido de la posición de memoria 197 ($00C5), que contiene el valor entero de la tecla que se acaba de pulsar. Si no se pulsa ninguna tecla cuando se ejecuta la lectura de esta posición (instrucción PEEK), se devuelve el valor 64. En el Apéndice C se muestran los valores numéricos de las teclas, los símbolos y sus caracteres equivalentes (CHR$). El siguiente ejemplo espera la pulsación de una tecla y convierte su código en la letra adecuada.

```
10 AA=PEEK(197):IF AA=64 THEN 10
20 BB$=CHR$(AA)
```

El teclado es tratado como una serie de interruptores organizados en una matriz de 8 filas y 8 columnas. La matriz del teclado es examinada por el KERNAL usando el chip CIA#1 (MOS 6526 Adaptador Complejo de Interface) para detectar si se ha cerrado algun interruptor. Se usan dos registros de la CIA para realizar la comprobación: el registro #0 en la posición 56320 ($DC00) para las columnas del teclado y el registro #1 en la posicion 56321 ($DC01) para las filas del teclado.
Los bits 0-7 de la posición 56320 corresponden a las columnas 0-7. Los bits 0-7 de la posición 56321 corresponden a las filas 0-7 del teclado. Leyendo estos registros, el KERNAL decodifica los interruptores cerrados en los correspondientes códigos de caracter (CHR$) de las teclas pulsadas.
8 columnas por 8 filas dan un total de 64 valores, o teclas, posibles. Sin embargo, si primero pulsa las teclas RVS, CTRL, C= o SHIFT, y luego escribe otro carácter, se generan valores adicionales. Esto sucede porque el KERNAL decodifica separadamente estas teclas y "recuerda" cuál ha sido pulsada. El resultado de la comprobación del teclado se almacena en la posición 197.
Los caracteres pueden también ser entrados directamente en el buffer haciendo POKE en las posiciones 631-640. Estos caracteres se usan cuando se hace otro POKE en la posición 198 con el número de caracteres introducidos en el buffer. Esta característica se puede usar para que se ejecuten automáticamente comandos en modo directo, imprimiéndolos en la pantalla, llenando el buffer de retornos de carro, y haciendo el POKE correspondiente en la posición 198. En el siguiente ejemplo, el programa se lista en la impresora continuando luego su ejecución.

```
10 PRINT CHR$(147)"PRINT#1:CLOSE1:GOTO50"
20 POKE 631,19:POKE 632,13:POKE 633,13:POKE 198,3
30 OPEN 1,4:CMD1:LIST
40 END
50 REM EL PROGRAMA CONTINUA A PARTIR DE AQUI
```

EDITOR DE PANTALLA

El EDITOR DE PANTALLA le provee de potentes y útiles facilidades para la edición de programas. Cuando se lista una sección de un programa en la pantalla, las teclas de control del cursor, así como otras teclas especiales le permiten corregir fácilmente los errores. Después de haber realizado los cambios necesarios en una línea de programa, pulse la tecla RETURN en cualquier lugar de la línea y el editor de pantalla leerá la línea lógica de 80 caracteres entera.

Entonces el texto es pasado al intérprete BASIC, que comprime las palabras reservadas y lo almacena en la memoria del programa. La línea editada reemplaza la vieja versión de la línea en la memoria. Se puede crear una copia adicional de cualquier línea cambiando su número y pulsando RETURN.

Si usa abreviaciones de palabras clave que causen que una línea exceda los 80 caracteres, los caracteres sobrantes se perderán en caso de editarla, puesto que el EDITOR lee sólo dos líneas físicas en la pantalla. Por este motivo tampoco se pueden entrar más de 80 caracteres en respuesta a un INPUT. Por todos estos motivos, la longitud de una línea de programa se limita en la práctica a 80 caracteres, no siendo aconsejable superarlos.

En determinadas condiciones, el EDITOR DE PANTALLA trata las teclas de control del cursor de distinto modo. Si el cursor se posiciona detrás de un número impar de comillas (") el editor entra en MODO COMILLAS.

En modo comillas, los datos se escriben igual, pero las teclas de control del cursor no lo mueven, sino que producen un carácter inverso que permanece en la línea. Lo mismo ocurre con las teclas de control de color. Esto le permite incluir controles de color o del cursor en sus programas, dentro de una cadena. Usted encontrará esta característica sumamente interesante, puesto que le da una potente herramienta de edición de datos. Cuando se imprime un texto, todos los controles de color y cursor se ejecutarán automáticamente, permitiéndole escribir en cualquier parte de la pantalla y en cualquiera de los colores disponibles. Un ejemplo del uso de los caracteres de control de cursor en una cadena es:

Usted escribe ->10 PRINT"A(R)(R)B(L)(L)(L)C(R)(R)D"REM (R)= CRSR DERECHA,(L)=CRSRIZQUIERDA

El ordenador imprime---> AC BD

La tecla DEL es la única no afectada por el modo comillas. Si se produce un error en modo comillas tiene varias opciones: borrar el testo hasta el error mediante DEL o pulsar RETURN y volver a la línea, con lo que se cancela el modo comillas y podrá editarla normalmente. Las teclas de control del cursor que puede usar en una cadena se muestran en la tabla 2-2.

Tabla 2-2. Caracteres de control del cursor en MODO COMILLAS

Tecla de control		Apariencia
CRSR arriba	CRSR	
CRSR abajo	CRSR	
CRSR izquierda	CRSR	
CRSR derecha	CRSR	
CLR	CLR/HOME	
HOME	CLR/HOME	S
INST	INST/DEL	

Cuando NO está en modo comillas, pulsando SHIFT e INST/DEL se desplazan los datos a la derecha del cursor para abrir un espacio en que insertar nuevos datos. El EDITOR pasa a MODO DE INSERCION hasta que el espacio abierto ha sido llenado.

En el modo de inserción los controles del cursor aparecen también como símbolos inversos. La única diferencia está en la tecla INST/DEL. La tecla DEL, que operaba normalmente en modo comillas, ahora presenta una T inversa. La tecla INST, que creaba un símbolo inverso en modo comillas, ahora actúa normalmente. Esto significa que se puede crear un PRINT conteniendo DELetes, lo que no se puede lograr en modo comillas. El modo de insercion se cancela pulsando RETURN, SHIFT y RETURN, o RUN/STOP y RESTORE. Por supuesto, también se cancela al rellenar todos los espacios insertados. Un ejemplo del uso de DEL puede ser:

10 PRINT"HELLO" DEL INST INST DEL DEL P"

(La secuencia anterior se mostrará como sigue al listarse:)

10 PRINT"HELP"

Cuando se ejecute (RUN) el ejemplo, se imprimirá HELP, puesto que las letras LO se han borrado antes de imprimir la P. El carácter DEL funciona igual al listarse, por lo que puede usarlo para "esconder" parte del listado de un programa. Sin embargo, la edición de una línea de estas características puede ser muy difícil, por no decir imposible.

Hay otros caracteres con funciones especiales, aunque no son accesibles fácilmente mediante el teclado. Para colocarlos entre comillas, debe dejar espacios vacíos en la línea, pulsar RETURN o SHIFT RETURN, volver a estos espacios, pulsar CTRL RVS/ON, para iniciar la impresión inversa, y escribir las teclas mostradas abajo:

FUNCION	ESCRIBA	APARECE
SHIFT RETURN	SHIFT M	
	N	N
Cambiar a minúsculas	SHIFT N	
Cambiar a mayúsculas		
Desactivar teclas de cambio de Mayúsc./Minúsc.		
Activar teclas de cambio de Mayúsc./Minúsc.		

El carácter SHIFT RETURN realiza un retorno de carro pero no finaliza la cadena. Esto trabaja igual en el listado, por lo que es prácticamente imposible editar estas líneas. Cuando se envía un listado a la impresora, el carácter "N inversa" coloca a la impresora en modo Mayúsculas/Minúsculas, y SHIFT "N inversa" produce la vuelta a Mayúsculas/gráficos.

Se pueden incluir caracteres en video inverso dentro de una cadena pulsando la tecla CTRL y RVS/ON (9) lo que causa la aparición de una R inversa dentro de las comillas. Todos los caracteres que sigan a este símbolo se imprimirán en video inverso (igual que un negativo fotográfico). Para finalizar este tipo de impresión pulse CTRL y RVS/OFF (0) lo que imprime un símbolo gráfico inverso. Los datos numéricos se pueden imprimir en video inverso colocando primero CHR$(18). Imprimiendo CHR$(146) o un retorno de carro se cancela la salida de video inverso.

CAPITULO 3

PROGRAMACION DE GRAFICOS EN EL COMMODORE 64

- Generalidades
- Posiciones de gráficos
- Modo carácter (standard)
- Caracteres programables
- Modo gráfico multicolor
- Modo extendido de color del fondo
- Gráficos "Bit Mapped"
- Modo Bit Map multicolor
- Scroll uniforme
- Sprites
- Otras características gráficas
- Programación de Sprites, otro modo

GENERALIDADES SOBRE GRAFICOS

Todas las habilidades gráficas del Commodore 64 provienen del Video Interface Chip 6567 (también conocido como VIC-II chip). Este chip ofrece una gran variedad de modos gráficos, incluyendo un display de 40 columnas por 25 líneas de texto, un display de alta resolución de 320 por 200 puntos, y SPRITES, pequeños objetos móviles con los que realizar juegos es fácil. Y por si fuera poco, muchos de los modos gráficos pueden mezclarse en la misma pantalla. Es posible, por ejemplo, definir la mitad superior de la pantalla en alta resolución, la mitad inferior utilizarla para texto, y los SPRITES se pueden combinar con las dos mitades. Hablaremos de los SPRITES más adelante. Primero veamos los otros modos gráficos.
El VIC-II chip posee los siguientes modos gráficos:

A) MODOS DE MOSTRAR CARACTERES

1) Modo carácter standard

a) Caracteres en ROM
b) Caracteres programables en RAM

2) Modo Carácter

a) Caracteres en ROM
b) Caracteres programables en RAM

3) Modo de color del fondo extendido

a) Caracteres en ROM
b) Caracteres programables en RAM

B) MODOS BIT MAP

1) Modo Bit Map standard
2) Modo Bit Map multicolor

C) SPRITES

1) Sprites standard
2) Sprites multicolores

POSICIONES DE GRAFICOS

Primero veamos alguna información general. La pantalla del Commodore 64 tiene 1000 posiciones de memoria. Normalmente, la pantalla se inicia en la posición 1024 ($0400 en HEXadecimal) y termina en la posición 2023. Cada una de estas posiciones tiene 8 bits. Esto significa que pueden contener cualquier número entero comprendido entre 0 y 255. Conectado con la memoria de pantalla hay otro grupo de 1000 posiciones llamado **MEMORIA DE COLOR** o **RAM DE COLOR**. Este grupo empieza en la posición 55296 ($D800 en HEX) y termina en la posición 56295. Cada posición de la RAM de color tiene 4 bits, lo que significa que puede contener cualquier entero entre 0 y 15. Como el Commodore 64 puede representar 16 colores distintos, esto funciona bien.
Hay 256 caracteres que se pueden mostrar al mismo tiempo. En la pantalla normal, cada una de las 1000 posiciones contiene el código que indica al VIC-II chip el carácter a mostrar en la posición que lo contiene.
Los distintos modos gráficos se seleccionan mediante los 47 registros de **CONTROL** que posee el VIC-II chip. Muchas de las funciones gráficas pueden ser controladas colocando mediante POKE los valores correctos en los registros adecuados. El VIC-II chip está colocado entre las posiciones 53248 ($D000 en HEX) y 53294 ($D02E en HEX).

SELECCION DE BANCO DE VIDEO

El VIC-II chip puede acceder a 16K. de memoria al mismo tiempo. Puesto que el Commodore 64 posee 64K. de memoria, usted debe ser hábil para lograr que el VIC-II chip acceda a la parte de memoria que le interesa. Esta es la forma de hacerlo: Hay 4 posibles bancos (o secciones) de 16K. cada uno. Todo lo que necesita es algún medio para controlar el bancó de 16K. al que el VIC-II debe acceder. De esta forma, el VIC-II chip puede acceder a los 64K. de memoria del Commodore 64. Los bits de **SELECCION DE BANCO** que le permiten acceder a las distintas secciones de memoria están colocados en el chip **ADAPTADOR COMPLEJO DE INTERFACE 6526 (CIA#2)**. Las instrucciones POKE y PEEK del BASIC (o sus versiones en código máquina) se usan para seleccionar el banco de memoria controlando los bits 0 y 1 del PORT A de la CIA#2 (posición 56576 ($DD00 HEX)). Estos dos bits deben ser inicializados para cambiar el banco. El siguiente ejemplo lo muestra:

```
POKE 56578,PEEK(56578)OR 3: REM INICIALIZACION
POKE 56576,(PEEK(56576)AND 252)OR A: REM CAMBIO DE BANCO
```

"A" debe ser uno de los siguientes valores:

VALOR DE A		BANCO	POSICION INICIAL	ACCESO DEL VIC-II CHIP
0	00	3	49152	($C000–$FFFF)*
1	01	2	32768	($8000–$BFFF)
2	10	1	16384	($4000–$7FFF)*
3	11	0	0	($0000–$3FFF) (VALOR NORMAL)

El concepto de los bancos de 16K. es sólo una parte de las cosas que hace el VIC-II chip. Usted debe vigilar siempre con que banco trabaja el VIC-II chip, ya que ello afecta a los caracteres, la posición de la pantalla, los sprites, etc. Al conectar el **Commodore 64, los bits 0 y 1 de la posición 56576 seleccionan el BANCO 0 ($0000-$3FFF).**

***NOTA: El juego de caracteres del Commodore 64 no es accesible por el VIC-II chip en los bancos 1 y 3. (Consulte la sección de memoria de caracteres)**

MEMORIA DE PANTALLA

La posición de la memoria de pantalla puede ser cambiada fácilmente con POKE del registro de control 53272 ($D018 HEX). Sin embargo, este registro también se usa para controlar el juego de caracteres empleado. Los cuatro bits SUPERIORES controlan la posición de la pantalla. Para moverla debe usar lo siguiente:

```
POKE 53272,(PEEK(53272)AND15)OR A
```

Donde "A" tiene uno de los siguientes valores:

A	BITS	POSICION*	
		DECIMAL	HEX
0	0000XXXX	0	$0000
16	0001XXXX	1024	$0400 (NORMAL)
32	0010XXXX	2048	$0800
48	0011XXXX	3072	$0C00
64	0100XXXX	4096	$1000
80	0101XXXX	5120	$1400
96	0110XXXX	6144	$1800
112	0111XXXX	7168	$1C00
128	1000XXXX	8192	$2000
144	1001XXXX	9216	$2400
160	1010XXXX	10240	$2800
176	1011XXXX	11264	$2C00
192	1100XXXX	12288	$3000
208	1101XXXX	13312	$3400
224	1110XXXX	14336	$3800
240	1111XXXX	15360	$3C00

*Recuerde que debe ser añadida la DIRECCION DE BANCO del VIC-II chip

MEMORIA DE COLOR

La memoria de color NO se puede mover. Está situada siempre en las posiciones 55296-56295 ($D800-$DBE7). La memoria de pantalla (las mil posiciones que empiezan en 1024) y la memoria de color se usan de distinta forma según el modo gráfico utilizado. Un dibujo creado en un modo con frecuencia se ve completamente distinto al ser mostrado en otro modo gráfico.

MEMORIA DE CARACTERES

La forma exacta en que el VIC-II obtiene la información sobre los caracteres e importante en la programación de gráficos. Normalmente, el chip encuentra la forma del carácter que desea imprimir en la ROM DEL GENERADOR DE CARACTERES. En este chip se almacenan los modelos de varias letras, números, signos de puntuación, y los otros símbolos presentes en el teclado.
Una de las características del Commodore 64 es la capacidad para usar modelos almacenados en la memoria RAM. Estos modelos en RAM son creados por usted, lo que significa que puede tener una gama casi infinita de símbolos para juegos, aplicaciones, etc.
Un juego de caracteres normal contiene 256 caracteres, y cada uno de ellos se define mediante 8 bytes de datos. Puesto que cada carácter ocupa 8 bytes, el juego completo de 256 caracteres ocupará 256*8=2K bytes de memoria.
El VIC-II chip puede acceder a 16K de memoria a la vez, por lo que tenemos 8 posibles posiciones para un juego de caracteres completo. Naturalmente, usted es libre de usar menos caracteres, pero el juego debe empezar en una de las 8 direcciones de inicio.
La posición de la memoria de caracteres se controla por 3 bits del registro de control del VIC-II chip en la posición 53272 ($D018 en HEX). Los bits 3,2, y 1 controlan en que bloque de 2K se encuentra el juego de caracteres. El bit 0 es ignorado. Recuerde que este es el mismo registro que determina donde se inicia la memoria de pantalla, por lo que no debe interferir con los bits de la memoria de pantalla.Para cambiar la posición de la memoria de caracteres debe usarse lo siguiente:

```
POKE 53272,(PEEK(53272)AND240"OR A
```

Donde A tiene uno de los siguiente valores:

VALOR DE A	BITS	POSICION DE LA MEMORIA DE CARACTERES*	
		DECIMAL	HEX
0	XXXX000X	0	$0000-$07FF
2	XXXX001X	2048	$0800-$0FFF
4	XXXX010X	4096	$1000-$17FF IMAGEN ROM EN BANCOS 0 Y 2 (NORMAL)
6	XXXX011X	6144	$1800-$1FFF IMAGEN ROM EN BANCOS 0 Y 2
8	XXXX100X	8192	$2000-$27FF
10	XXXX101X	10240	$2800-$2FFF
12	XXXX110X	12288	$3000-$37FF
14	XXXX111X	14336	$3800-$3FFF

**Recuerde añadir la dirección del BANCO*

La IMAGEN ROM reseñadas en la tabla anterior se refiere al generador de caracteres ROM. Este generador aparece en lugar de la RAM en las posiciones especificadas del banco 0. También aparece en las posiciones RAM 36864-40959 ($9000-$9FFF) en el banco 2. Puesto que el VIC-II solo puede acceder a 16K al mismo tiempo, los caracteres en ROM aparecen en el bloque de 16K que es controlado por el VIC-II chip. Por esto, el sistema se diseñó de forma que el VIC-II piense que los caracteres en ROM están en 4096-8191 ($1000-$1FFF) cuando se selecciona el banco 0, y en 36864-40959 ($9000-$9FFF) cuando se selecciona el banco 2, a pesar de que los caracteres en ROM se encuentran en realidad en 53248-57343 ($D000-$DFFF). Esto solo se utiliza por el VIC-II para tomar los datos de los caracteres, por lo que estas zonas de RAM pueden ser usadas para programas, otros datos,etc., al igual que cualquier otra zona de memoria RAM.

> NOTA: Si estos caracteres en ROM se interfieren con sus propios graficos, seleccione los bancos 1 o 3, en los que el VIC-II no tiene acceso a los mismos.

La posición y contenido del juego de caracteres en ROM es la siguiente:

	DIRECCION		ACCESO	CONTENIDO
BLOQUE	DECIMAL	HEX	DEL VIC-II	
0	53248	D000-D1FF	1000-11FF	Caract. en mayúsculas
	53760	D200-D3FF	D200-13FF	Caract. gráficos
	54272	D400-D5FF	1400-15FF	Mayúsculas invertidas
	54784	D600-D7FF	1600-17FF	Gráficos invertidos
1	55296	D800-D9FF	1800-19FF	Caract. en minúsculas
	55808	DA00-DBFF	1A00-1BFF	Mayúsculas y gráficos
	56320	DC00-DDFF	1C00-1DFF	Minúsculas invertidas
	56832	DE00-DFFF	1E00 1FFF	Minúsculas y gráficos invertidos

Los lectores avispados pueden haberse dado cuenta de lo siguiente. Las posiciones ocupadas por el generador de caracteres en ROM son las mismas que algunas ocupadas por los registros de control del VIC-II chip. Esto es posible porque no ocupan las mismas posiciones al mismo tiempo.
Cuando el VIC-II necesita acceder a los datos de los caracteres, la ROM que los contiene es activada. Ello produce una imagen de esta ROM en el banco de 16K al que tiene acceso el VIC-II. De este modo, el área está ocupada por los registros de control de E/S, y la ROM de caracteres sólo está disponible para el VIC-II.
Sin embargo, usted puede necesitar los datos contenidos en la ROM de caracteres si usa caracteres programables y desea copiar algunos de los contenidos en la ROM. En este caso, usted debe desactivar los registros de E/S, activar la ROM del generador de caracteres y entonces copiar los caracteres que desee. Una vez haya terminado, debe activar de nuevo los registros de E/S. Durante el proceso de copiado (cuando las E/S están desactivadas) no se permite ninguna interrupción. Esto ocurre porque los registros de E/S son necesarios para realizar una interrupción. Si usted olvida esto e interrumpe el proceso pueden suceder cosas realmente extrañas. El teclado no debe ser leído durante el proceso. Para desconectar el teclado y otras interrupciones normales debe utilizarse lo siguiente:


```
POKE 56334,PEEK(56334)AND 254     (DESACTIVA INTERRUPCIONES)
```

Una vez finalice el proceso de copiado y esté listo para seguir con su programa, debe activar el teclado con lo siguiente:

```
POKE 56334,PEEK(56334)OR 1        (ACTIVA INTERRUPCIONES)
```

El siguiente POKE desactiva las E/S y activa la ROM del generador de caracteres:

```
POKE 1,PEEK(1)AND 251
```

La ROM del generador de caracteres está ahora en las posiciones desde 53248 hasta 57343 ($D000-$DFFF).
Para colocar de nuevo las E/S a partir de $D000 para el funcionamiento normal use el siguiente POKE:

```
POKE1,PEEK(1)OR 4
```

MODO CARACTER (STANDARD)

El modo carácter standard es el modo en que se encuentra el Commodore 64 al ser conectado. Es el modo en que usted programará normalmente.
Los carácteres se pueden obtener de la memoria ROM o RAM, aunque normalmente se tomarán de la ROM. Si usted necesita símbolos especiales para un programa determinado deberá definir su forma en la memoria RAM, y ordenar al VIC-II chip que busque la información sobre los carácteres en la RAM en lugar de en la ROM. Esto se explica con más detalle en la próxima sección.
Para mostrar los caracteres en color en su pantalla, el VIC-II accede a la memoria de pantalla para determinar el código de carácter que corresponde a una posición determinada de la pantalla. Al mismo tiempo, accede a la memoria de color para determinar el color en que se mostrará el carácter. El código de carácter es traducido por el VIC-II en la dirección de inicio del bloque de 8 bytes que contienen la forma del carácter. Estos 8 bytes se encuentran en la memoria de caracteres.
La traducción no es muy complicada, pero se combinan una serie de factores para generar la dirección adecuada. En primer lugar, el código de carácter que se usa para hacer un POKE en la memoria de pantalla se multiplica por 8. Después se añade la dirección de inicio de la memoria de carácteres (vea la sección MEMORIA DE CARACTERES). Después, los bits de selección de banco se tienen en cuenta para añadir a la dirección resultante. La siguiente fórmula le ilustra el procedimiento:

DIRECCION DEL CARACTER = CODIGO DE PANTALLA*8 + (JUEGO DE CARACTERES*2048)+(BANCO*16384)

DEFINICION DE LOS CARACTERES

Cada carácter está formado por una malla de 8 por 8 puntos, en la que cada punto puede ser visible o invisible. Las formas de los carácteres del Commodore 64 están

almacenadas en la ROM del generador de caracteres. Los caracteres se almacenan en juegos de ocho bytes para cada carácter, donde cada byte representa una línea de 8 puntos, y cada bit representa un punto. Un bit con el contenido 0 significa que el correspondiente punto está apagado (invisible). Si el bit contiene 1 el punto será visible.
La memoria de carácteres en ROM empieza en la posición 53248 (cuando las E/S están desconectadas). Los primeros 8 bytes desde la posición 53248 ($D00) a 53255 ($D007) contienen la forma del signo @, que tiene el código de carácter cero en la memoria de pantalla. Los próximos 8 bytes (53256-53263) ($D008-$D00F), contienen la información para formar la letra A.

IMAGEN	BINARIO	PEEK
**	00011000	24
****	00111100	60
** **	01100110	102
******	01111110	126
** **	01100110	102
** **	01100110	102
** **	01100110	102
	00000000	0

Cada juego de caracteres completo ocupa 2K (2048 bytes) de memoria, 8 bytes por carácter y 256 caracteres por juego. Puesto que hay dos juegos de caracteres, uno para las mayúsculas y gráficos y otro para mayúsculas y minúsculas, el generador de caracteres ROM ocupa un total de 4K de memoria.

CARACTERES PROGRAMABLES

Puesto que los caracteres se almacenan en ROM, parece que no hay forma de cambiarlos para crear nuestros propios caracteres. Sin embargo, la posición de memoria que indica al VIC-II donde debe buscar las formas de los caracteres es un registro programable que puede ser cambiado para que apunte a varias secciones de memoria. Cambiando el puntero de la memoria de caracteres, el juego de caracteres puede ser programado para cualquier necesidad.
Si desea que su juego de caracteres se almacene en RAM, hay una serie de cosas MUY IMPORTANTES que debe saber para poder crear un programa con juego de caracteres propios. Además, hay otros dos importantes puntos que debe conocer para crear sus propios juegos de caracteres:

1) Este es un proceso de todo o nada. Generalmente, si usa su propio juego de caracteres diciendo al VIC II que busque la información sobre los mismos en el área que preparó en la RAM, el juego de caracteres standard del Commodore 64 no estará disponible para usted. Para resolver esto, usted puede copiar letras, números, etc. para usarlos desde su propio generador de caracteres en RAM. Usted puede escoger los que desee, sólo los que necesite, y colocarlos en orden en la RAM.

2) Su juego de caracteres utiliza un espacio de memoria que en principio está reservado a sus programas BASIC. Por supuesto, con 38K disponibles, la mayoría de las aplicaciones no van a tener problemas.

ATENCION: Usted debe proteger cuidadosamente su generador de caracteres para que el programa BASIC no se escriba encima, ya que usa también la RAM.

Hay dos posiciones de inicio del generador de caracteres que **NO se pueden utilizar con BASIC: posición 0 y posición 2048.** La primera no se puede usar porque el sistema operativo almacena información importante en la página cero. La segunda tampoco puede ser usada porque en esta dirección es donde empieza el programa en BASIC. Sin embargo, hay otras seis direcciones de inicio para almacenar su generador de caracteres.
El mejor lugar para colocar su juego de caracteres para usar con el BASIC mientras experimenta es la dirección de inicio 12288 ($3000 en HEX). Esto se logra haciendo un POKE en la dirección 53272 para colocar 12 en los cuatro bits más bajos de la misma. Pruebe el siguiente POKE:

```
POKE 53272,(PEEK(53272)AND 240)+12
```

Inmediatamente todas las letras de la pantalla se convierten en una serie de manchas. Esto ocurre porque en la dirección 12288 y siguientes no hay ningún juego de caracteres, sinó unicamente bytes con un valor aleatorio. Restablezca la normalidad en su Commodore 64 pulsando simultáneamente las teclas RUN STOP y RESTORE.
Ahora vamos a iniciar la creación de caracteres gráficos. Para proteger su juego de caracteres del BASIC, debe reducir la cantidad de memoria destinada al mismo. El total de memoria en su ordenador será el mismo, pero el BASIC no podrá usar parte de esta memoria. Teclee:

```
PRINT FRE(0)-(SGN(FRE(0))<0)*65535
```

El resultado de la fórmula es la cantidad de memoria no utilizada. Ahora teclee lo siguiente:

```
POKE 52,48:POKE 56,48:CLR
```

Ahora teclee:

```
PRINT FRE(0)-(SGN(FRE(0))<0)*65535
```

Ve el cambio? El BASIC piensa que dispone de menos memoria para trabajar. La memoria escamoteada al BASIC la podrá utilizar para su generador de caracteres, sin que el BASIC la invada.
El próximo paso es colocar sus caracteres en la RAM. Cuando empieza, sólo hay datos aleatorios en las posiciones 12288 y siguientes. Usted debe colocar la información sobre los caracteres en la RAM (en el mismo estilo que la contenida en la ROM) para que pueda usarla el VIC-II
El siguiente programa traslada 64 caracteres de la ROM a su juego de caracteres en RAM:

```
5 PRINTCHR$(142):REM CONECTA LA CAJA ALTA
10 POKE52,48:POKE56,48:CLR:REM RESERVA MEMORIA PARA LOS CARACTERES
20 POKE56334,PEEK(56334)AND254:REM DESCONECTA LA EXPLORACION DEL TECLADO
30 POKE1,PEEK(1)AND251:REM CONECTA EL CARACTER
40 FORI=0TO511:POKEI+12288,PEEK(I+53248):NEXT
50 POKE1,PEEK(1)OR4:REM CONECTA I/O
60 POKE56334,PEEK(56334)OR1:REM CONECTA LA EXPLORACION DEL TECLADO
70 END
READY.
```

Ahora haga un POKE en la posición 53272 con (PEEK(53272)AND 240) + 12. Nada sucede, ¿correcto? Bien, casi nada. El Commodore 64 está ahora buscando la información sobre los caracteres en su RAM, en lugar de la ROM. Pero como que ha copiado los caracteres de la ROM exactamente igual, no puede haber diferencias... todavía.

Ahora puede usted cambiar facilmente la forma de los caracteres. Borre la pantalla y teclee el signo @. Mueva el cursor unas líneas hacia abajo y escriba:

```
FOR I = 12288 TO 12288+7:POKE I,255-PEEK(I):NEXT
```

¡Usted acaba de crear un signo @ en color invertido!

> NOTA Los signos en color invertido son caracteres con los bits que contienen su información también invertidos.

Ahora mueva el cursor hasta el principio de la línea y pulse **RETURN** de nuevo para que el carácter se vuelva a presentar normal. Mirando la tabla de códigos de pantalla usted puede deducir en que parte de RAM se almacena cada carácter. Sólo recuerde que cada carácter ocupa ocho posiciones de memoria. He aquí algunos ejemplos de inicios:

CARACTER	CODIGO PANTALLA	POSICION DE INICIO EN RAM
@	0	12288
A	1	12296
!	33	12552
>	62	12784

Recuerde que sólo se han copiado los primeros 64 caracteres. Cualquier otro debe copiarse también si usted lo necesita.

¿Cómo obtener el carácter número 154, Z invertida? Bien, usted puede hacerlo utilizando el procedimiento descrito en la página anterior, copiando el juego de caracteres invertidos de la ROM o bien colocar el carácter que precisa en lugar de uno de los caracteres innecesarios que almacena en RAM.

Suponga que no va a necesitar el signo >. Entonces reemplácelo por la Z invertida. Teclee esto:


```
FOR I=0 TO 7: POKE 12784+I,255-PEEK(I+12496):NEXT
```

Ahora escriba el signo >. Aparece como una Z invertida. No importa cuántas veces escriba el signo >, siempre aparecerá como Z invertida. (Este cambio es en realidad una ilusión. Aunque el signo > aparezca como Z invertida, actuará en un programa como >). Pruebe alguna cosa que precise el signo >. Verá como trabaja bien, sólo tiene una apariencia extraña.

Una rápida revisión: Usted puede ahora copiar caracteres de la ROM a la RAM. Puede elegir los caracteres que precise. Sólo hay una manera de comprender perfectamente el modo de operar: ¡practique con sus propios caracteres!

¿Recuerda cómo se almacenan los caracteres en ROM?. Cada carácter se almacena en un grupo de 8 bytes. El modelo de los bytes controla la forma del carácter. Si toma 8 bytes, uno encima del otro, y escribe cada byte en forma de ocho dígitos binarios (bits), forma una malla de 8 por 8, con el modelo del carácter. Si el bit está a 1, habrá un punto en la posición. Con el valor 0, hay un espacio en la posición. Para crear sus propios caracteres, debe confeccionar una tabla similar en la memoria del ordenador. Escriba NEW y teclee el siguiente programa:

```
10 FOR I = 12448 TO 12455:READ A:POKE I,A:NEXT
20 DATA 60,66,165,129,165,153,66,60
```

Ahora escriba RUN. El programa reemplaza la letra T por una cara sonriente. Escriba unas cuantas T para ver la cara. Cada uno de los números en la instrucción DATA de la línea 20 es una línea del carácter que representa la cara sonriente. La matriz para dicha cara es la siguiente:

	7	6	5	4	3	2	1	0	BINARIO	DECIMAL
FILA 0			*	*	*	*			00111100	60
1		*					*		01000010	66
2	*		*			*		*	10100101	165
3	*							*	10000001	129
4	*		*			*		*	10100101	165
5	*			*	*			*	10011001	153
6		*					*		01000010	66
FILA 7			*	*	*	*			00111100	60

La hoja de trabajo para caracteres programables (Figura 3-1) le ayuda a diseñar sus propios caracteres. En la hoja hay una matriz de 8 por 8, con filas y columnas numeradas. (Si toma cada fila como un número binario, los números de la parte superior indican el valor de cada bit. Cada número es una potencia de 2. El bit de la izquierda tiene el valor de 128 –2 elevado a la 7 potencia– el próximo es 64 (2 a la sexta), y así hasta llegar al bit de la derecha (bit 0), que es igual a 1 (o 2 elevado a la potencia 0).

Coloque una X en cada posición donde quiera un punto para su carácter. Cuando lo tenga listo debe crear la instrucción DATA para su carácter.

	7	6	5	4	3	2	1	0
0								
1								
2								
3								
4								
5								
6								
7								

Fig 3–1. Hoja de trabajo para Caracteres Programables

Empiece por la primera fila. Anote el valor de cada bit donde haya colocado una X. (El valor se obtiene del número en la parte superior de la matriz, como se ha explicado antes). Sume los valores de todos los bits de una fila y escriba el resultado a la derecha de la misma. Este será el número que deberá colocar en la lista DATA para dibujar esta línea.

Repita el proceso con las otras filas (1–7). Una vez finalizado tendrá 8 números comprendidos entre 0 y 255. Si alguno de los números no está dentro de este rango, repase la suma, ya que los números deben estar entre estos valores para ser correctos. Si tiene menos de 8 números habrá olvidado una fila. Repase. Recuerde que las filas con valor 0 son tan importantes como cualquier otra.

Reemplace los números de la instrucción DATA de la línea 20 por los números que ha calculado usted, y ejecute el programa. Después escriba la letra T. ¡Cada vez que lo haga, usted verá su propio carácter!

Si no le gusta como ha quedado el carácter, simplemente debe cambiar los números de la lista DATA y volver a ejecutar el programa hasta que el carácter creado le satisfaga plenamente.

> **NOTA:** Para obtener los mejores resultados, haga todas las líneas verticales de sus caracteres de 2 bits de ancho. Esto ayuda a prevenir la distorsión de color de sus caracteres cuando se muestran en una pantalla de televisión.

A continuación se muestra un programa de ejemplo que usa caracteres programables standard:

```
10 REM *EJEMPLO 1*
20 REM CREANDO CARACTERES PROGRAMABLES
31 POKE56334,PEEK(56334)AND254:POKE1,PEEK(1)AND251:REM DESCONECTAR KB Y I/O
35 FORI=0TO63:REM EL RANGO DEL CARACTER ES COPIADO DEL ROM
36 FORJ=0TO7:REM COPIA LOS 8 BYTES POR CARACTER
37 POKE12288+I*8+J,PEEK(53248+I*8+J):REM COPIA UN BYTE
38 NEXTJ:NEXTI:REM VA AL SIGUIENTE BYTE O CARACTER
39 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1:REM CONECTA I/O Y KB
40 POKE53272,(PEEK(53273)AND240)+12:REM COLOCA EL PUNTERO DE CARACTERES
45 REM A LA MEMORIA 12288
60 FORCAR=60TO63:REM PROGRAMA CARACTERES 60 AL 63
80 FORBYTE=0TO7:REM RECORRE TODOS LOS 8 BYTES DE UN CARACTER
100 READ NUMERO:REM LEE 1/8TH DEL CARACTER DEL DATO
120 POKE12288+(8*CAR)+BYTE,NUMERO:REM ALMACENA LOS DATOS EN LA MEMORIA
140 NEXTBYTE:NEXTCAR:REM VA AL SIGUIENTE BYTE,CHAR
150 PRINTCHR$(147)TAB(255)CHR$(60);
155 PRINTCHR$(61)TAB(55)CHR$(62)CHR$(63)
160 REM LA LINEA 150 PONE LOS NUEVOS CARACTERES DEFINIDOS EN LA PANTALLA
170 GETA$:REM ESPERA QUE EL USUARIO PULSE UNA TECLA
180 IFA$="" THENGOTO170:REM SI NO SE PULSA NINGUNA TECLA,ESPERE
190 POKE53272,21:REM VUELVE A LOS CARACTERES NORMALES
200 DATA4,6,7,5,7,7,3,3:REM DATOS PARA EL CARACTER 60
210 DATA 32,96,224,160,224,224,192,192:REM DATOS PARA EL CARACTER 61
220 DATA7,7,7,31,31,95,143,127:REM DATOS PARA EL CARACTER 62
230 DATA 224,224,224,248,248,248,240,224:REM DATOS PARA EL CARACTER 63
240 END
```

MULTICOLOR MODO GRAFICO

Los gráficos standard de alta resolución le permiten controlar puntos muy pequeños en la pantalla. Cada punto en la memoria de caracteres puede tener dos valores: 1 para activarse y 0 para desactivarse. Cuando el punto esta desactivado, se muestra un espacio del color de la pantalla. Si el punto está activado, aparecerá del color que se haya elegido en la memoria de color. Usando los gráficos standard de alta resolución, todos los puntos de la malla (8*8) que forman un carácter pueden ser del color de la pantalla o del color del carácter. En algunos casos este límite en la resolución de color puede ser un problema. Por ejemplo, dos líneas juntas de dos colores distintos.

El modo multicolor ofrece la solución a este problema. Cada punto en el modo multicolor puede ser de cuatro colores: color de la pantalla (registro de color # 0), del

color del registro de color # 1, del colór del registro de color # 2, o del color del carácter. Sin embargo, se debe sacrificar la resolución horizontal, ya que cadá punto en modo multicolor es el doble de ancho que uno en alta resolución standard. Esta mínima pérdida de resolución queda compensada por las características extras del modo multicolor.

MULTICOLOR MODO BIT

Para activar el modo multicolor, coloque el bit 4 del registro 53270 ($D016) del VIC-II a 1, usando el siguiente POKE:

```
POKE 53270,PEEK(53270)OR 16
```

Para desactivar el modo multicolor, debe colocar este bit a 0 de la siguiente forma:

```
POKE 53270,PEEK(53270)AND 239
```

El modo multicolor se activa y desactiva para cada posición determinada de la pantalla, lo que permite mezclarlo con gráficos standard de alta resolución. Esto se controla por el bit 3 de la memoria de color. La memoria de color se inicia en la posición 55296 ($D800). Si el número contenido en una posición de la memoria de color es menor de 8 (0–7), el correspondiente espacio en la pantalla tendrá alta resolución standard, en el color que haya elegido (0–7). Si el número de la memoria de color es mayor de 7 (8–15), el espacio correspondiente aparecerá en modo multicolor. Haciendo un POKE en una posición de la memoria de color, usted puede cambiar el color del carácter en la posición correspondiente de la pantalla. Si el número del POKE está comprendido entre 0 y 7 obtendrá los colores normales de los caracteres. Si el número está entre 8 y 15, el espacio correspondiente de la pantalla se mostrará en modo multicolor. En otras palabras, colocando a 1 el bit 3 de la memoria de color, se activa el modo multicolor. Colocando este bit a 0, el modo multicolor se desactiva.

Una vez activado el modo multicolor en un espacio de la pantalla, los bits del carácter determinan el color en que se mostrará cada punto. Por ejemplo, aquí tiene el dibujo de la letra A, y los correspondientes bits que lo forman:

```
IMAGEN     BINARIO

   **      00011000
  *****    00111100
 **  **    01100110
 ******    01111110
 **  **    01100110
 **  **    01100110
 **  **    01100110
           00000000
```


En modo normal o de alta resolución, se muestra el color de la pantalla en cada punto cuyo bit esté a 0, y el color del carácter se muestra en los puntos cuyos bits estén a 1. El modo multicolor usa los bits agrupados en parejas, de la siguiente forma:

```
IMAGEN      BINARIO
AABB        00 01 10 00
CCCC        00 11 11 00
AABBAABB    01 10 01 10
AACCCCBB    01 11 11 10
AABBAABB    01 10 01 10
AABBAABB    01 10 01 10
AABBAABB    01 10 01 10
            00 00 00 00
```

En la imagen anterior, los espacios marcados AA se dibujan en el color 1, los espacios marcados BB usan el color 2, y los espacios marcados CC usan el color del carácter. Las parejas de bits determinan esto, de acuerdo con la siguiente tabla:

PAREJA DE BITS	REGISTRO DE COLOR	POSICION
00	Registro 0 (pantalla)	53281 ($D021)
01	Registro 1	53282 ($D022)
10	Registro 2	53283 (D023)
11	Color especificado por los tres bits más bajos de la memoria de color	RAM de color

Escriba NEW y entre el siguiente programa de demostración:

```
100 POKE53281,1:REM PONE EN BLANCO EL COLOR DEL FONDO 0
110 POKE53282,3:REM PONE EN CIANO EL COLOR DEL FONDO 1
120 POKE53283,8:REM PONE EN NARANJA EL COLOR DEL FONDO 2
130 POKE53270,PEEK(53270)OR16:REM CONECTA EL MODO MULTICOLOR
140 C=13*4096+8*256:REM COLOCA LA C AL PUNTO DEL COLOR DE MEMORIA
150 PRINTCHR$(147)"AAAAAAAAAA"
160 FORL=0TO9
170 POKEC+L,8
180 NEXT
```

El color de la pantalla es blanco, el color del carácter, negro, un registro de color contiene el color cyan y el otro el color naranja.

Usted no coloca realmente códigos de color en el espacio para el color del carácter, sinó que usa referencias a los registros asociados con estos colores. Esto le permi-

te utilizar algunos trucos. Cambiando el valor de uno de los registros de color, todos los puntos que se mostraban en el color cambiado, también cambiarán. Además, los colores de la pantalla y del fondo se pueden cambiar instantáneamente.

He aquí un ejemplo para cambiar el color del registro # 1:

```
100 POKE53270,PEEK(53270)OR16:REM CONECTA EL MODO MULTICOLOR
110 PRINTCHR$(147)CHR$(18);

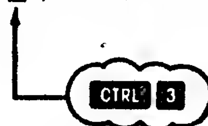
          C= 8
120 PRINT"  ";:REM TECLEA C=& PARA EL FONDO NEGRO MULTICOLOR O NARANJA
130 FORL=1TO22:PRINTCHR$(65);:NEXT
135 FORT=1TO500:NEXT

          CTRL 7
140 PRINT"  ";:REM TECLEA CTRL&7 PARA EL CAMBIO A AZUL
145 FORT=1TO500:NEXT

          CTRL 1
150 PRINT"  PULSE UNA TECLA"
160 GETA$:IFA$=""THEN160
170 X=INT(RND(1)*16)
180 POKE 53282,X
190 GOTO 160
```

Usando la tecla C= y las teclas de color los caracteres se pueden cambiar a cualquier color, incluidos los caracteres multicolor. Por ejemplo, escriba lo siguiente:

POKE 53270,PEEK(53270)OR16:PRINT " " ;REM ROJO CLARO/MULTICOLOR ROJO



La palabra READY y todo lo que escriba se mostrará en modo multicolor. Otro control de color puede volver al texto normal.

A continuación se muestra un programa que utiliza caracteres programables multicolores.

```
10 REM * EJEMPLO 2 *
20 REM CREANDO CARACTERES PROGRAMABLES MULTICOLORES
31 POKE56334,PEEK(56334)AND254:POKE1,PEEK(1)AND251
35 FORI=0TO63:REM COPIA EL RANGO DEL CARACTER DE LA ROM
36 FORJ=0TO7:REM COPIA TODOS LOS 8 BYTES POR CARACTER
37 POKE12288+I*8+J,PEEK(53248+I*8+J):REM COPIA UN BYTE
38 NEXTJ,I:REM VA AL SIGUIENTE BYTE O CARACTER
39 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1:REM CONECTA I/O Y KB
40 POKE 53272,(PEEK(53272)AND240)+12:REM PONE EL CARACTER PUNTERO EN
4  REM LA  MEMORIA 12288
50    53270,PEEK(53270)OR16
51 POKE53281,0:REM PONE EN NEGRO EL COLOR DEL FONDO 0
52 POKE53  ,2:REM PONE EN ROJO EL COLOR DEL FONDO 1
53 POKE53283,7:REM PONE EN AMARILLO EL COLOR DEL FONDO 3
60 FORCAR=60 TO 63:REM PROGRAMA LOS CARACTERES DEL 60 AL 63
 0 FORBYTE=0TO7:REM HACE LOS 8 BYTES DE UN CARACTER
1 0 READNUMERO:REM LEE EL 1/8TH DEL DATO DEL CARACTER
120 POKE12288+(8*CAR)+BYTE,NUMERO:REM ALMACENA LOS DATOS EN MEMORIA
140 NEXTBYTE,CAR  SHIFT CLR/HOME
150 PRINT" "TAB(255)CHR$(60)CHR$(61)TAB(55)CHR$(62)CHR$(63)
160 REM LA LINEA 150 PONE EN LA PANTALLA LOS NUEVOS CARACTERES DEFINIDOS
170 GETA$:REM ESPERA QUE EL USUARIO PULSE UNA TECLA
180 IFA$=""THEN170:REM SI NO SE PULSA NINGUNA TECLA,ESPERA
190 POKE53272,21:POKE53270,PEEK(53270)AND239:REM VUELVE AL CARACTER NORMAL
200 DATA129,37,21,29,93,85,85,85:REM DATOS PARA EL CARACTER 60
210 DATA66,72,84,116,117,85,85,85:REM DATOS PARA EL CARACTER 61
220 DATA87,87,85,21,8,8,40,0:REM DATOS PARA EL CARACTER 62
230 DATA213,213,85,84,32,32,40,0:REM DATOS PARA EL CARACTER 63
240 END
```

MODO EXTENDIDO DEL COLOR DEL FONDO

Este modo le permite controlar el color del fondo de cada carácter individual. Por ejemplo, usted puede mostrar un carácter azul sobre fondo amarillo en una pantalla blanca.

Hay 4 registros disponibles para el modo extendido del color del fondo. Cada uno de estos registros puede contener cualquiera de los 16 colores disponibles.

La memoria de color se usa para contener el color de los puntos en este modo, al igual que en el modo carácter standard.

El modo extendido de color del fondo limita el número de caracteres disponibles. Cuando está activado sólo los primeros 64 caracteres del generador ROM (o los 64 primeros caracteres de su propio juego) se pueden usar. Esto sucede porqué dos de los bits del código de carácter se usan para determinar el color del fondo. Así trabaja este modo:
El código de carácter (el número con el que se hacen los POKES en pantalla) de la letra "A" es 1. Con el modo extendido del color del fondo activado, si hace un POKE de 1 en la pantalla aparecerá una A. Si intenta el POKE con un valor de 65, debería aparecer una A invertida. Esto NO sucede en este modo. Usted obtendrá una A normal, pero con el color del fondo distinto. La siguiente tabla le muestra los distintos códigos:

CODIGO DE CARACTER			REGISTRO DE COLOR	
VALOR	BIT 7	BIT 6	NUMERO	DIRECCION
0–63	0	0	0	53281 ($D021)
64–127	0	1	1	53282 ($D022)
128–191	1	0	2	53283 ($D023)
192–255	1	1	3	53284 ($D024)

El modo extendido de color del fondo se activa colocando a 1 el bit 6 del registro 53265 ($D011) del VIC-II. El siguiente POKE lo activa.

POKE 53265,PEEK(53265)OR 64

El modo extendido de color del fondo se desactiva colocando el bit 6 del registro del VIC-II localizado en la dirección 53265 ($D011) A 0. La siguiente línea lo hace:

POKE 53265,PEEK(53265)AND 191

GRAFICOS "BIT MAPPED"

Cuando escriba juegos, gráficos para aplicaciones de negocios u otros tipos de programas, tarde o temprano necesitará displays de alta resolución.
El Commodore 64 ha sido diseñado para poder realizarlo: la alta resolución está disponible "mapeando" la pantalla. "Bit Mapping" es el método en el cual cada punto de resolución de la pantalla se asigna a un bit en la memoria. Si el bit en la memoria está a 1, el punto correspondiente estará encendido. Si el bit está a 0 el punto estará apagado.
El diseño de gráficos de alta resolución tiene un par de desventajas, por lo que no se usa todo el tiempo. La primera es que el mapeado de la pantalla ocupa una gran cantidad de memoria. Esto ocurre porque cada punto tiene un bit que lo controla. Usted necesita un bit por cada punto (o un byte por cada 8 puntos). Puesto que cada carácter tiene 8*8 puntos, y hay 25 líneas de 40 caracteres cada una, la resolución es de 320*200 para toda la pantalla. Esto da un total de 64000 puntos, cada uno de los cuales requiere un bit. En otras palabras, son necesarios 8000 bytes para mapear la pantalla.

Generalmente, las operaciones de alta resolución están formadas por rutinas simples que se repiten constantemente. Desafortunadamente, estos procesos son bastante lentos si las rutinas están escritas en BASIC. Sin embargo, las rutinas cortas y repetitivas se prestan mucho a ser programadas en código máquina. La solución es escribir sus programas de alta resolución en cógigo máquina, o confeccionar rutinas en este lenguaje y llamarlas desde su programa BASIC mediante la instrucción SYS. De esta forma se combina la facilidad de programar en BASIC con la rapidez del código máquina. El cartucho VSP está también disponible para añadir comandos de alta resolución al BASIC de su Commodore 64.
Todos los ejemplos que se ofrecen en esta sección están escritos en BASIC para una fácil comprensión. Ahora pasemos a los detalles técnicos.

"BIT MAPPING" es una de las más populares técnicas para gráficos de las utilizadas en el mundo de los ordenadores. Esta técnica se usa para crear detallados dibujos. Básicamente, cuando el Commodore 64 entra en este modo, muestra directamente en la pantalla una sección de 8K de memoria. Estando en el modo bit map, puede controlar directamente y encender o apagar cada punto de la pantalla.

Hay dos tipos de "bit mapping" disponibles en el Commodore 64. Estos son:

1) Standard (alta resolución) "Bit Map" (320*200)
2) Multicolor "Bit Map" (160*200)

Los dos son similares en características a los de caracteres: gran resolución con pocos colores contra una resolución más pequeña pero con posibilidad de elegir un mayor número de colores.

MODO "BIT MAP" DE ALTA RESOLUCION STANDARD

El modo standard de alta resolución le proporciona una resolución de 320 puntos horizontales por 200 verticales, con la elección de 2 colores en cada sección de 8*8. Para activar este modo debe colocar a 1 el bit 5 del registro del VIC-II 53265 ($D011), de la siguiente forma:

POKE 53265,PEEK(53265)OR 32

Para desactivar el modo "Bit Map" coloque a 0 el bit 5 del registro de control 53265 ($D011), de la siguiente forma:

POKE 53265,PEEK(53265)AND 223

Antes de entrar en más detalles sobre este modo hay algo más que debe saber, y es donde se encuentra el área de "bit map"

COMO TRABAJA

Si recuerda la sección de CARACTERES PROGRAMABLES deberá recordar cómo se forma el modelo de un carácter almacenado en RAM y cómo se cambia para adaptarlo a sus necesidades. Usted puede cambiar un solo punto de un carácter determinado cambiando el valor de una posición de memoria. Esta es la base del "bit-mapping". La pantalla entera se rellena de caracteres programables, y usted realiza

los cambios directamente en la memoria de los caracteres programables de donde extraen los modelos.
Cada una de las posiciones de la memoria de pantalla que se usaban para controlar qué carácter se tenía que mostrar sirven ahora para almacenar información sobre el color. Por ejemplo, haciendo un POKE en la posición 1024 con el valor 1 aparecía una "A" en la esquina superior izquierda de la pantalla. Ahora, la dirección 1024 controla el color de este espacio de la pantalla.
Los colores en el modo Bit Map no se toman de la memoria de color, como sucedía en el modo carácter. Ahora, los colores se extraen de la memoria de pantalla. Los 4 bits más altos de la memoria de pantalla controlan el color de los puntos cuyos bits están a 1. Los 4 bits más bajos de la memoria de pantalla controlan el color de los puntos cuyos bits están a 0. Cada posición de la memoria de pantalla controla el color de la correspondiente malla de 8*8.

EJEMPLO: Escriba lo siguiente:

```
5 BASE=2*4096:POKE53272,PEEK(53272)OR8:REM PONE EL MAPA DE BIT EN 8192
10 POKE53265,PEEK(53265)OR32:REM ENTRA EL MODO DEL MAPA DE BIT
```

Ahora ejecute (RUN) el programa.
La pantalla se llena de porquería, ¿no? Al igual que con la pantalla normal, usted debe borrar la pantalla de alta resolución antes de trabajar con ella. Desafortunadamente, CLR no funciona en este caso. Usted debe limpiar la sección de memoria que usará para sus caracteres programables. Pulse las teclas **RUN/STOP** y **RESTORE** simultáneamente. Después, añada las siguientes líneas a su programa para borrar la pantalla de alta resolución:

```
20 FORI=BASETOBASE+7999:POKEI,0:NEXT:REM LIMPIA EL MAPA DE BIT
30 FORI=1024TO2023:POKEI,3:NEXT:REM COLOCA EL COLOR EN CIANO Y NEGRO
```

Ahora ejecute el programa otra vez. Usted debe ver la pantalla limpia, después el color cyan debe cubrir la pantalla. Lo que necesita ahora es encender y apagar los puntos deseados en el display de alta resolución.

Para ACTIVAR un punto o DESACTIVAR un punto debe saber como encontrar el bit correcto en la memoria de caracteres para ponerlo a 1. En otras palabras, debe saber el carácter a cambiar, la fila donde se encuentra y el bit de la fila que debe ser cambiado. Usted necesita una fórmula para calcular esto.
Usaremos X e Y para situar las posiciones horizontal y vertical de un punto. El punto situado en X=0 e Y=0 será el de la esquina superior izquierda de la pantalla. Los puntos hacia la derecha tendrán valores de X más altos, y los puntos hacia abajo tendrán un valor de Y más alto. La mejor forma de usar el bit mapping es dibujar un display de bit map similar a éste:


```
0 ·········· X ·········· 319
·
Y
·
·
199 ·························
```

Cada punto tiene un valor de X e Y que determinan una coordenada. En este formato es fácil controlar cada punto de la pantalla.

Sin embargo, lo que usted tiene es algo parecido a esto:

------	BYTE 0	BYTE 8	BYTE 16	BYTE 24	BYTE 312
	BYTE 1	BYTE 9	.	.	BYTE 313
	BYTE 2	BYTE 10	.	.	BYTE 314
	BYTE 3	BYTE 11	.	.	BYTE 315
	BYTE 4	BYTE 12	.	.	BYTE 316
	BYTE 5	BYTE 13	.	.	BYTE 317
	BYTE 6	BYTE 14	.	.	BYTE 318
------	BYTE 7	BYTE 15	.	.	BYTE 319
------	BYTE 320	BYTE 328	BYTE 336	BYTE 344	BYTE 632
	BYTE 321	BYTE 329	.	.	BYTE 633
	BYTE 322	BYTE 330	.	.	BYTE 634
	BYTE 323	BYTE 331	.	.	BYTE 635
	BYTE 324	BYTE 332	.	.	BYTE 636
	BYTE 325	BYTE 333	.	.	BYTE 637
	BYTE 326	BYTE 334	.	.	BYTE 638
------	BYTE 327	BYTE 335	.	.	BYTE 639

Los caracteres programables que forman el bit map se organizan en 25 líneas de 40 columnas cada una. Si bien éste es un buen método para la organización de texto,

para el bit map resulta algo complicado. (Hay una buena razón para utilizar este método. Vea la sección MODOS MIXTOS).
La siguiente fórmula le permite obtener fácilmente el control de cada punto de la pantalla:
El inicio de la memoria del display se conoce como BASE. El número de línea (de 0 a 24) de su punto es:

RO=INT(Y/8) (Hay 320 bytes por línea)

La posicion del carácter para esta línea (de 0 a 39) es:

CH=INT (X/8) (Hay 8 bytes por carácter)

La línea para el carácter situado en esta posición es (de 0 a 7):

LI=Y AND 7

El bit de este byte es:

BI=7-(X AND 7)

Ahora, coloquemos estas fórmulas juntas. El byte donde se encuentra el punto que queremos cambiar se localiza mediante la fórmula:

BY=BASE + RO*320+CH*8+LI

Para activar cualquier bit de la malla con coordenadas (X,Y), use esta línea:

POKE BY,PEEK(BY) OR2 ↑ BI

Ahora coloque estos cálculos en un programa. En el siguiente ejemplo el Commodore 64 traza una curva senoidal:

```
50 FORX=0TO319:REM UNA ONDA LLENARA LA PANTALLA
60 Y=INT(90+80*SIN(X/10))
70 CH=INT(X/8)
80 RO=INT(Y/8)
85 LH=YAND7
90 BY=BASE+RO*320+8*CH+LH
100 BI=7-(XAND7)
110 POKEBY,PEEK(BY)OR(2↑BI)
120 NEXTX
125 POKE1024,16
130 GOTO 130
```

El cálculo de la línea 60 cambia los valores para la función seno del rango de +1 a -1 al rango de 10 a 170. Las líneas 70 a 100 calculan el carácter, línea, byte y bit afectados, usando las fórmulas explicadas anteriormente. La línea 125 indica que el programa ha terminado cambiando el color de la esquina superior izquierda de la pantalla. La línea 130 hace que el programa entre en un bucle sin fin. Cuando haya visto los resultados del programa, pulse RUN/STOP y RESTORE simultáneamente.

Como ejemplo adicional, puede modificar el programa de la curva sinusoide para que dibuje un semicírculo. He aquí las líneas a escribir para realizar estos cambios:

```
50 FORX=0TO160:REM HACE LA MITAD DE LA PANTALLA
55 Y1=100+SQR(160*X-X*X)
56 Y2=100-SQR(160*X-X*X)
60 FORY=Y1TOY2STEPY1-Y2
70 CH=INT(X/8)
80 RO=INT(Y/8)
85 LH=YAND7
90 BY=BASE+RO*320+8*CH+LH
100 BI=7-(XAND7)
110 POKEBY,PEEK(BY)OR(2↑BI)
114 NEXT
```

Esto crea un semicírculo en el área de alta resolución de la pantalla.

> **ATENCION**: Las variables BASIC pueden invadir el espacio reservado para alta resolución. Si necesita más memoria para su programa, deberá mover la dirección de inicio del BASIC pasada el área de alta resolución, o cambiar de lugar dicha área. Este problema NO sucede programando en lenguaje máquina. UNICAMENTE puede suceder cuando se escriben programas en BASIC.

MODO "BIT MAP" MULTICOLOR

Al igual que en el modo carácter, el modo bit map multicolor le permite mostrar hasta cuatro colores distintos en cada sección de 8 por 8 del bit map. Y también al igual que en el modo carácter, la resolución horizontal se sacrifica (de 320 puntos a 160 puntos).
El modo bit map multicolor usa 8K de memoria para realizar el "mapeado de pantalla" (bit map). Usted puede seleccionar los colores desde (1) el registro de color 0, (el color del fondo de la pantalla), (2) la matriz de video (los cuatro bits más altos dan un color; los 4 bits más bajos otro), y (3) de la memoria de color.
El modo bit map multicolor se activa colocando a uno el bit 5 de la posición 53265 ($D011) y el bit 4 de la posición 53270 ($D016). El siguiente POKE hace esto:

POKE53265,PEEK(53265)OR 32: POKE53270,PEEK(53270)OR 16

El modo bit map multicolor se desactiva colocando a 0 el bit 5 de la posición 53265 ($D011) y el bit 4 de la posición 53270 ($D016). El siguiente POKE hace esto:

POKE 53265,PEEK(53265)AND 223: POKE 53270,PEEK(53270)AND 239

Al igual que en el modo de alta resolución standard, hay una correspondencia de uno a uno entre la sección de 8K usada para el display de alta resolución y lo que se muestra en la pantalla. Sin embargo, los puntos horizontales tienen dos bits de ancho. Cada 2 bits de la memoria reservada para alta resolución forman 1 punto, que puede ser de 4 colores según la disposición de los dos bits.

BITS	LA INFORMACION DE COLOR PROCEDE DE
00	Registro de color #0 (color de la pantalla)
01	Los 4 bits más altos de la memoria de pantalla
10	Los 4 bits más bajos de la memoria de pantalla
11	Memoria de color (4 bits)

SCROLL UNIFORME

(Deslizamientos del contenido de la pantalla)

El VIC-II soporta el scroll uniforme en todas direcciones. El scroll uniforme es el movimiento de toda la pantalla punto a punto en una dirección determinada. La pantalla se puede mover arriba, abajo, a la izquierda o a la derecha. Se usa para introducir uniformemente nueva información en la pantalla, mientras que uniformemente desaparecen los caracteres de la cara opuesta de la misma.

Entre las características del VIC-II se encuentran las de poder colocar la pantalla en cualquiera de 8 posiciones horizontales y 8 verticales. El posicionado es controlado por los registros de scroll del VIC-II. Este chip posee también un modo de 38 columnas, y uno de 24 líneas. La disminución de la pantalla se usa para que pueda colocar los datos que precisan del scroll.

A continuación se muestran los pasos a seguir para lograr el SCROLL UNIFORME:

1) Reduzca la pantalla (el borde se amplía)
2) Coloque el registro de scroll al máximo (o al mínimo, dependiendo de la dirección del scroll)
3) Coloque los datos en la parte apropiada de la pantalla (cubierta)
4) Incremente (o disminuya) el registro de scroll hasta que alcance el valor máximo. (o mínimo)
5) En este punto, use su rutina en lenguaje máquina para mover toda la pantalla un carácter en la dirección del scroll.
6) Vuelva al paso 2.

Para entrar en el modo de 38 columnas, debe poner a cero el bit 3 de la posición 53270 ($D016). El siguiente POKE lo realiza:

POKE 53270,PEEK(53270)AND 247

Para volver al modo de 40 columnas el bit 3 de la posición 53270 ($D016) debe colocarse a 1, mediante el siguiente POKE:

POKE 53270,PEEK(53270)OR 8

Para entrar en el modo de 24 líneas, el bit 3 de la posición 53265 ($D011) debe ponerse a 0. El siguiente POKE lo hace:

POKE 53265,PEEK(53265)AND 247

Para volver al modo de 25 líneas el bit 3 de la posición 53265 ($D011) debe ponerse a 1, mediante el siguiente POKE:

POKE 53265,PEEK(53265)OR 8

Cuando se hace un scroll en la dirección X debe colocarse el VIC-II en el modo de 38 columnas. Esto permite colocar en su sitio los datos afectados por el scroll. Cuando este sea hacia la IZQUIERDA, los nuevos datos deben colocarse a la derecha. Si el scroll es hacia la DERECHA, los datos se colocarán a la izquierda. Recuerde que la pantalla sigue teniendo 40 columnas, lo que sucede es que sólo son visibles 38.

Al hacer un scroll en la dirección Y, es necesario que el VIC-II se coloque en el modo de 24 líneas. Al hacer un scroll hacia ARRIBA, coloque los datos en la última línea. Cuando el scroll sea hacia ABAJO, coloque los datos en la primera línea. Al contrario que en el scroll X, en el que se cubrían areas en cada lado de la pantalla, hay una sola área cubierta en el scroll Y. Si el registro de scroll Y está a cero, se cubre la primera línea. Cuando el registro de scroll Y está a 7, se cubre la última línea.

Para el scroll en la dirección X, el registro está situado en los bits 2 a 0 del registro de control del VIC-II en la posición 53270 ($D016). Como siempre, es importante que la modificación afecte únicamente a estos bits. El siguiente POKE lo hace:

POKE 53270,(PEEK(53270)AND 248)+X

donde X es la posición de la pantalla de 0 a 7.

Para hacer scroll en la dirección Y, el registro está situado en los bits 2 a 0 de la posición 53265 ($D011). Es importante que la modificación afecte únicamente a estos bits. El siguiente POKE hace esto:

POKE 53265,(PEEK(53265)AND 248)+Y

donde Y es la posición de la pantalla de 0 a 7.

Para hacer un scroll de texto en la pantalla desde abajo, debe colocar los 3 bits menos significativos de la posición 53265 de 0 a 7, colocar los datos en la línea cubierta de la parte inferior de la pantalla, y repetir el proceso. Para hacer un scroll de izquierda a derecha, debe pasar los 3 bits menos significativos de la posición 53270 de 0 a 7, PRINT o POKE una columna de datos en la columna 0 de la pantalla, y repetir el proceso.

Si usted pasa los bits de scroll por −1, su texto se moverá en dirección opuesta.

EJEMPLO: Scroll de texto en la parte inferior de la pantalla:

```
10 POKE53265,PEEK(53265)AND247:REM VA LA EL MODO DE LA COLUMNA 24
20 PRINTCHR$(147):REM LIMPIA LA PANTALLA
30 FORX=1TO24:PRINTCHR$(17);:NEXT:REM MUEVE EL CURSOR ABAJO DE TODO
40 POKE53265,(PEEK(53265)AND248)+7:PRINT:REM POSICIONA PRIMER DESPLAZAMIENTO
50 PRINT"   HOLA";
60 FORP=6TO0STEP-1
70 POKE53265,(PEEK(53265)AND248)+P
80 FORX=1TO50:NEXT:REM DEMORA EL BUCLE
90 NEXT:GOTO40
```

SPRITES

Un Sprite es un tipo especial de carácter programable que puede ser situado en cualquier lugar de la pantalla. Los Sprites son mantenidos directamente por el VIC-II chip. Para realizar un Sprite usted debe decidir la forma que quiere que tenga, el color apropiado, y el lugar de la pantalla en que quiere situarlo. El VIC-II se encarga del resto. Los Sprites pueden ser de cualquier color de los dieciséis disponibles. Los Sprites se pueden usar conjuntamente con CUALQUIER otro modo gráfico: bit map, carácter, multicolor, etc., y siempre mantendrán su definición y forma, con cualquiera de ellos. Los Sprites poseen su propia definición de color, su modo (alta resolución o multicolor), y su forma.
El VIC-II puede mantener automáticamente hasta 8 Sprites al mismo tiempo. Se pueden mostrar más Sprites usando las técnicas de interrupción de barrido.
Las características de los Sprites son:

1) Tamaño: 24 puntos (horizontal) por 21 puntos (vertical)
2) Control de color individual para cada Sprite.
3) Modo Sprite Multicolor.
4) Magnificación (2X) en horizontal, vertical, o ambas direcciones.
5) Selección de la prioridad de los Sprites con la pantalla
6) Prioridad fija de los Sprites entre sí.
7) Detección de colisiones entre Sprites.
8) Detección de colisiones entre Sprites y la pantalla

Estas características especiales permiten programar fácilmente muchos video/juegos populares. Puesto que los Sprites son mantenidos por el hardware, es posible escribir en BASIC juegos de alta calidad.
Hay 8 Sprites soportados directamente por el VIC-II. Están numerados de 0 a 7. Cada uno de los Sprites tiene su propio registro de forma, registros de posición y registro de color, y también poseen sus propios bits para ser activados y detectar colisiones.

DEFINIR UN SPRITE

Los Sprites se definen del mismo modo que los caracteres programables. Sin embargo, puesto que los Sprites son de mayor tamaño, son necesarios más bytes para almacenar su forma. Un Sprite tiene 24 por 21 puntos, o 504 puntos. Esto significa que se precisan 63 bytes (504/8 bits) para definir un Sprite. Los 63 bytes se organizan en 21 líneas de 3 bytes cada una. La definición de un Sprite se asemeja a lo siguiente:

```
BYTE 0     BYTE 1     BYTE 2
BYTE 3     BYTE 4     BYTE 5
BYTE 6     BYTE 7     BYTE 8
  ..         ..         ..
  ..         ..         ..
  ..         ..         ..
BYTE 60    BYTE 61    BYTE 62
```


NUMERO DE COLUMNA	BIT	VALORES (ON = 1 × VAL)	LINEA 0	LINEA 1	LINEA 2	LINEA 3	LINEA 4	LINEA 5	LINEA 6	LINEA 7	LINEA 8	LINEA 9	LINEA 10	LINEA 11	LINEA 12	LINEA 13	LINEA 14	LINEA 15	LINEA 16	LINEA 17	LINEA 18	LINEA 19
00	7	125																				
01	6	64																				
02	5	32																				
03	4	16																				
04	3	8																				
05	2	4																				
06	1	2																				
07	0	1																				
08	7	128																				
09	6	64																				
10	5	32																				
11	4	16																				
12	3	8																				
13	2	4																				
14	1	2																				
15	0	1																				
16	7	128																				
17	6	64																				
18	5	32																				
19	4	16																				
20	3	8																				
21	2	4																				
22	1	2																				
23	0	1																				

Figura 3-2. Tabla de definición de Sprites.

Otra forma de ver cómo se crea un Sprite es mirando la tabla de definición de Sprites a nivel de bits. Puede ser parecida a la de la figura 3-2.
En los Sprites standard (alta resolución), cada bit con valor 1 se muestra en el color del Sprite. Los bits con valor 0 son transparentes y muestran el color existente debajo del mismo. Esto es similar a los caracteres standard.
Los Sprites multicolores son similares a los caracteres multicolores. La resolución horizontal es reducida para obtener colores extras. La resolución de un Sprite multicolor es de 12 puntos horizontales por 21 verticales. Cada punto horizontal es el doble de ancho, pero el número de colores disponibles por Sprite se incrementa hasta 4.

PUNTEROS DE LOS SPRITES

Aunque cada Sprite ocupa 63 bytes para su definición, se precisa un byte más para colocar al final del grupo de 63. Cada Sprite, pues, ocupa 64 bytes. Esto facilita los cálculos acerca de las posiciones de memoria que ocupa cada Sprite, ya que 64, además de un número, es una potencia binaria.
Cada uno de los Sprites posée un byte asociado llamado PUNTERO DE SPRITE. Este puntero controla la posición de la definición de cada sprite en la memoria. Estos 8 bytes están colocados siempre en las últimas 8 posiciones de memoria de la zona de 1K correspondiente a la memoria de pantalla. Normalmente, en el Commodore 64, esto significa que los ocho bytes empiezan en la posición 2040 ($07F8). Sin embargo, si usted mueve la pantalla, los punteros de los Sprites también cambiarán.
Cada puntero de Sprite puede contener un número entre 0 y 255. Este número indica la posición de la definición para su Sprite asociado. Puesto que cada definición ocupa 64 bytes, esto significa que el puntero puede indicar cualquiera de las posiciones del bloque de 16K al que el VIC-II tiene acceso (puesto que 256*64=16K).
Si el puntero de Sprite #0, en la posición 2040, contiene el número 14, por ejemplo, el Sprite 0 se mostrará con la forma contenida en los 64 bytes a partir de la posición 14*64 = 896 (esta zona está situada en el buffer del cassette). La siguiente fórmula aclara esta cuestión.

POSICION = (BANCO*16384) + (VALOR DEL PUNTERO DE SPRITE*64)

Donde BANCO es la zona de 16K de memoria a la que accede el VIC-II, y que puede tener un valor de 0 a 3.
La fórmula anterior da como resultado la posición de inicio de los 64 bytes que forman el bloque de definición de un Sprite.
Cuando el VIC-II accede al BANCO 0 o al BANCO 2, hay una IMAGEN ROM del juego de caracteres presente en algunas posiciones, como se mencionó antes. Las definiciones de los Sprites NO se pueden colocar aquí. Si por alguna razón necesita más de 128 definiciones de Sprites, utilice uno de los bancos sin IMAGEN ROM (bancos 1 o 3).

ACTIVAR UN SPRITE

El registro de control 53269 ($D015) se conoce como el registro ACTIVADOR DE SPRITES. Cada uno de los Sprites tiene un bit en este registro que indica si el mismo está activado o desactivado. El registro se podría ver de la siguiente forma:

53269 ($D015) 7 6 5 4 3 2 1 0

Para activar el Sprite 1, por ejemplo, es necesario colocar el valor 1 en el correspondiente bit. El siguiente POKE lo realiza:

```
POKE 53269,PEEK(53269)OR 2
```

Una sentencia más general podría ser la siguiente:

```
POKE 53269,PEEK(53269)OR (2 ↑ SN)
```

Donde SN es el número del Sprite, de 0 a 7.

> **NOTA**: Un Sprite debe ser activado antes de poder ser visualizado.

DESACTIVAR UN SPRITE

Un Sprite se desactiva colocando a cero su bit en el registro de control del VIC-II 53269 ($D015). El siguiente POKE lo hace:

```
POKE 53269, PEEK(53269)AND (255-2 ↑ SN)
```

donde SN es el número del Sprite (de 0 a 7).

COLORES

Un Sprite puede ser de cualquiera de los 16 colores generados por el VIC-II. Cada uno de los Sprites tiene su propio registro de color. Estas son las posiciones de memoria de los registros de color:

POSICION		DESCRIPCION
53287	($D027)	REGISTRO DE COLOR SPRITE 0
53288	($D028)	REGISTRO DE COLOR SPRITE 1
53289	($D029)	REGISTRO DE COLOR SPRITE 2
53290	($D02A)	REGISTRO DE COLOR SPRITE 3
53291	($D02B)	REGISTRO DE COLOR SPRITE 4
53292	($D02C)	REGISTRO DE COLOR SPRITE 5
53293	($D02D)	REGISTRO DE COLOR SPRITE 6
53294	($D02E)	REGISTRO DE COLOR SPRITE 7

Todos los puntos de un Sprite se muestran en el color contenido en su registro de color. El resto del Sprite es transparente, y muestra lo que hay detrás del mismo.

MODO MULTICOLOR

El modo multicolor le permite usar cuatro colores distintos en cada Sprite. Sin embargo, al igual que en los otros modos multicolores, la resolución horizontal se reduce a la mitad. En otras palabras, cuando trabaja en el modo Sprite multicolor (al igual que en el modo carácter multicolor), en lugar de 24 puntos horizontales por

Sprite, hay 12 pares de puntos. Cada par de puntos es llamado PAREJA DE BITS. Piense que cada pareja de bits (pareja de puntos) es un solo punto en su Sprite, y la información de los dos bits le permite escoger el color deseado. La tabla siguiente le indica los valores que debe tener cada pareja de bits para obtener un punto del color deseado para su Sprite:

PAR DE BITS	DESCRIPCION
00	TRANSPARENTE, COLOR DE LA PANTALLA
01	REGISTRO DE SPRITE MULTICOLOR #0 (53285) ($D025)
10	REGISTRO DE COLOR DEL SPRITE
11	REGISTRO DE SPRITE MULTICOLOR #1 (53286) ($D026)

COLOCAR UN SPRITE EN MODO MULTICOLOR

Para colocar un Sprite en modo multicolor, debe activar el registro de control del VIC-II 53276 ($D01C). El siguiente POKE lo hace:

POKE 53276,PEEK(53276)OR (2 ↑ SN)

donde SN es el número del Sprite (de 0 a 7).
Para colocar un Sprite fuera del modo multicolor, debe desactivar el registro de control del VIC-II en la posición 53276 ($D01C). El siguiente POKE lo hace:

POKE 53276,PEEK(53276) AND (255–2 ↑ SN)

donde SN es el número del Sprite (de 0 a 7).

SPRITES EXPANDIDOS

El VIC-II chip tiene la capacidad de expandir un Sprite en dirección vertical, horizontal, o ambas a la vez. Un Sprite expandido tiene los puntos que lo forman el doble de anchos o de altos. La resolución no se incrementa... el Sprite es sólo mayor.
Para expandir un Sprite en dirección horizontal, el bit correspondiente del registro de control 53277 ($D01D) debe colocarse a 1. El siguiente POKE expande un Sprite en la dirección X:

POKE 53277,PEEK(53277)OR (2 ↑ SN)

donde SN es el número del Sprite (de 0 a 7).
Para que un Sprite vuelva a su ancho normal, el bit correspondiente del registro del VIC-II en la posición 53277 ($D01D) debe desactivarse (ponerse a 0). El siguiente POKE "encoge" un Sprite en la dirección X:

POKE 53277,PEEK(53277)AND (255–2 ↑ SN)

donde SN es el número de Sprite (de 0 a 7).
Para expandir un Sprite en dirección vertical, el bit correspondiente del registro de control del VIC-II en la posición 53271 ($D017) debe colocarse a 1. El siguiente POKE expande un Sprite en la dirección Y:

POKE 53271,PEEK(53271)OR (2 ↑ SN)

donde SN es el número de Sprite (de 0 a 7).

Para que un Sprite vuelva a su altura normal, el bit correspondiente en el registro de control de la posición 53271 ($D017) debe colocarse a 0. El siguiente POKE "encoge" un Sprite en la dirección Y:

POKE 53271,PEEK(53271)AND (255–2 ↑ SN)

donde SN es el número de Sprite (de 0 a 7).

POSICIONADO DE SPRITES

Una vez haya diseñado un Sprite, usted deseará que se mueva por la pantalla. Para lograrlo, su Commodore 64 usa tres registros de posición:

1) REGISTRO DE POSICION DE SPRITE X
2) REGISTRO DE POSICION DE SPRITE Y
3) REGISTRO DEL BIT MAS SIGNIFICATIVO DE LA POSICION X

Cada Sprite posee un registro de posición X, un registro de posición Y, y un bit en el registro del bit más significativo de la posición X. Esto le permite situar los Sprites en la pantalla con gran precision. Puede colocar un Sprite en 512 posibles posiciones X y en 256 posibles posiciones Y.
Los registros de posición X e Y trabajan juntos, en parejas, como un equipo. Las posiciones de los registros X e Y se muestran en el mapa de memoria como sigue: Primero el registro X para el Sprite 0, después el registro Y para el Sprite 0. Después viene el registro X para el Sprite 1, el registro Y para el Sprite 1, etc. Después de los dieciséis registros X e Y viene el bit más significativo de la posición X (X MSB) localizado en su propio registro.
Después viene el registro X para el Sprite 1, el registro Y para el Sprite 1, etc. Después de los dieciséis registros X e Y viene el bit más significativo de la posición X (X MSB) localizado en su propio registro.
La siguiente tabla muestra las posiciones de los registros de posición para cada Sprite. Usted puede situar los valores correctos en cada posición mediante POKES.

POSICION		DESCRIPCION
DECIMAL	HEX	
53248	($D000)	REGISTRO DE POSICION X SPRITE 0
53249	($D001)	REGISTRO DE POSICION Y SPRITE 0
53250	($D002)	REGISTRO DE POSICION X SPRITE 1
53251	($D003)	REGISTRO DE POSICION Y SPRITE 1
53252	($D004)	REGISTRO DE POSICION X SPRITE 2
53253	($D005)	REGISTRO DE POSICION Y SPRITE 2
53254	($D006)	REGISTRO DE POSICION X SPRITE 3
53255	($D007)	REGISTRO DE POSICION Y SPRITE 3
53256	($D008)	REGISTRO DE POSICION X SPRITE 4
53257	($D009)	REGISTRO DE POSICION Y SPRITE 4
53258	($D00A)	REGISTRO DE POSICION X SPRITE 5
53259	($D00B)	REGISTRO DE POSICION Y SPRITE 5
53260	($D00C)	REGISTRO DE POSICION X SPRITE 6
53261	($D00D)	REGISTRO DE POSICION Y SPRITE 6
53262	($D00E)	REGISTRO DE POSICION X SPRITE 7
53263	($D00F)	REGISTRO DE POSICION Y SPRITE 7
53264	($D010)	REGISTRO DE SPRITE X MSB (Bit más significativo de la posición X.

La posición de un Sprite es calculada desde la esquina superior izquierda del área de 24 por 21 puntos destinada al mismo. No importa el número de puntos que active en un Sprite. Incluso en el caso de que sólo se use un punto en el Sprite, y desee situarlo en el centro de la pantalla, debe calcular la posición partiendo de la esquina superior izquierda del área destinada al mismo.

POSICIONADO VERTICAL

Puesto que el posicionado horizontal es algo más complicado que el vertical, hablaremos primero de este último.
Hay 200 posiciones distintas en dirección vertical (Y) en las que se muestra el Sprite en la pantalla de televisión. Los registros de posición Y para los Sprites pueden contener numeros de 0 a 255. Esto significa que usted tiene más que suficientes valores en el registro para mover un Sprite arriba y abajo. Usted puede necesitar que un Sprite entre y salga uniformemente de la pantalla. Para lograr esto son necesarios más de 200 valores.
El primer valor del registro —en la parte superior de la pantalla— y en la dirección Y para un Sprite sin expandir es 30. Para un Sprite expandido en la dirección Y debe ser 9 (puesto que cada punto es el doble de alto y la posición sigue siendo calculada a partir de la esquina superior izquierda del Sprite).
El primer valor en que el Sprite está totalmente dentro de la pantalla (expandido o no) es 50.
El último valor de Y en que un Sprite sin expandir está totalmente en la pantalla es 229. El último valor en el caso de un Sprite expandido en la dirección Y, mostrándose entero en la pantalla, es 208.
El primer valor de Y en que un Sprite está completamente fuera de la pantalla es 250.

EJEMPLO:

SHIFT CLR/HOME

```
10 PRINT"♥":REM LIMPIA LA PANTALLA
20 POKE2040,13:REM COGE EL DATO DEL SPRITE 0 DEL BLOQUE 13
30 FORI=0TO62:POKE832+I,129:NEXT:REM COLOCA EL DATO DEL SPRITE EN EL BLOQUE
35 REM 13 (13*64=832)
40 V=53248:REM COLOCA EL PRINCIPIO DEL CHIP DEL VIDEO
50 POKEV+21,1:REM FACILITA EL SPRITE 1
60 POKEV+39,1:REM COLOCA EL COLOR DEL SPRITE 0
70 POKEV+1,100:REM COLOCA LA POSICION Y DEL SPRITE 0
80 POKEV+16,0:POKEV,100:REM COLOCA LA POSICION X DEL SPRITE 0
```

POSICIONADO HORIZONTAL

El posicionado horizontal es más complicado porque hay más de 256 posiciones en que colocar el Sprite. Esto significa que es preciso un bit extra, o noveno bit, para controlar el posicionado horizontal. Añadiendo este noveno bit el Sprite tiene ahora 512 posibles posiciones en la dirección X (izq/der.) Esto da más posiciones para el Sprite de las necesarias para que vaya de un lado a otro de la pantalla. Cada Sprite puede tener una posición de 0 a 511. Sin embargo, sólo en los valores entre 24 y 343 es visible en la pantalla. Si la posición X del Sprite es superior a 255 (a la derecha de la pantalla) el bit correspondiente del registro MSB (MOST SIGNIFICANT BIT (bit más significativo)) de la posición X debe ser puesto a 1.

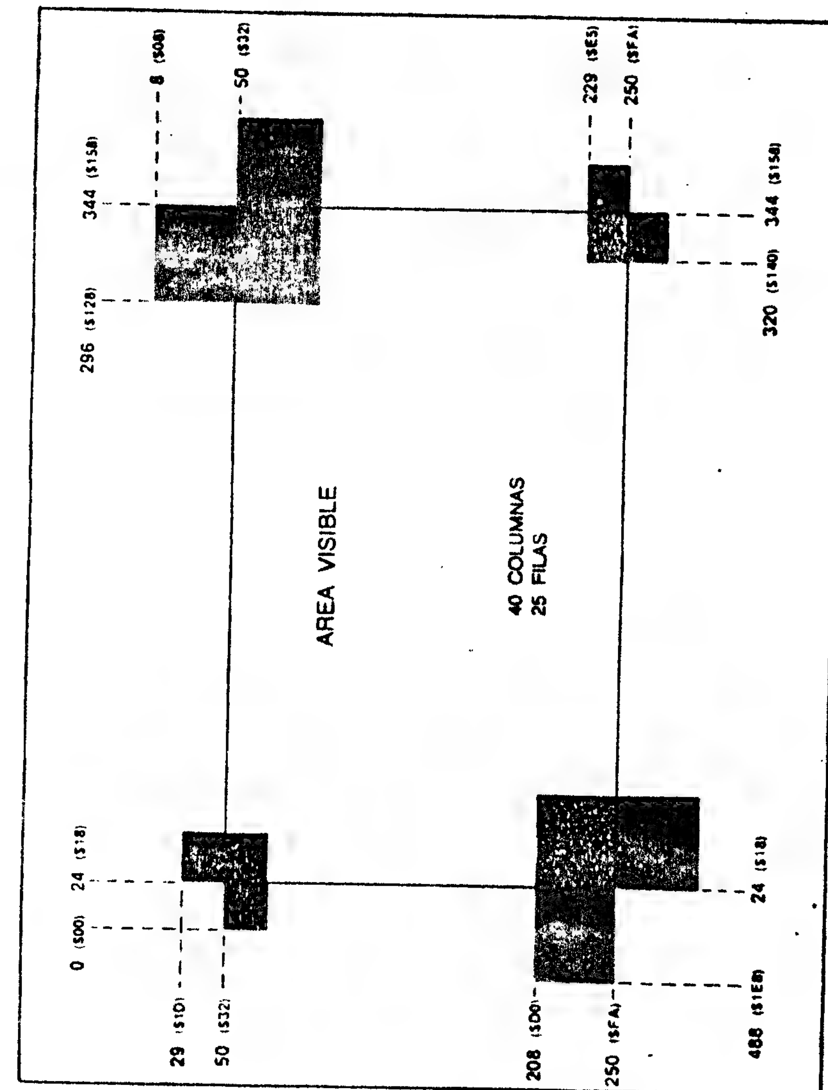
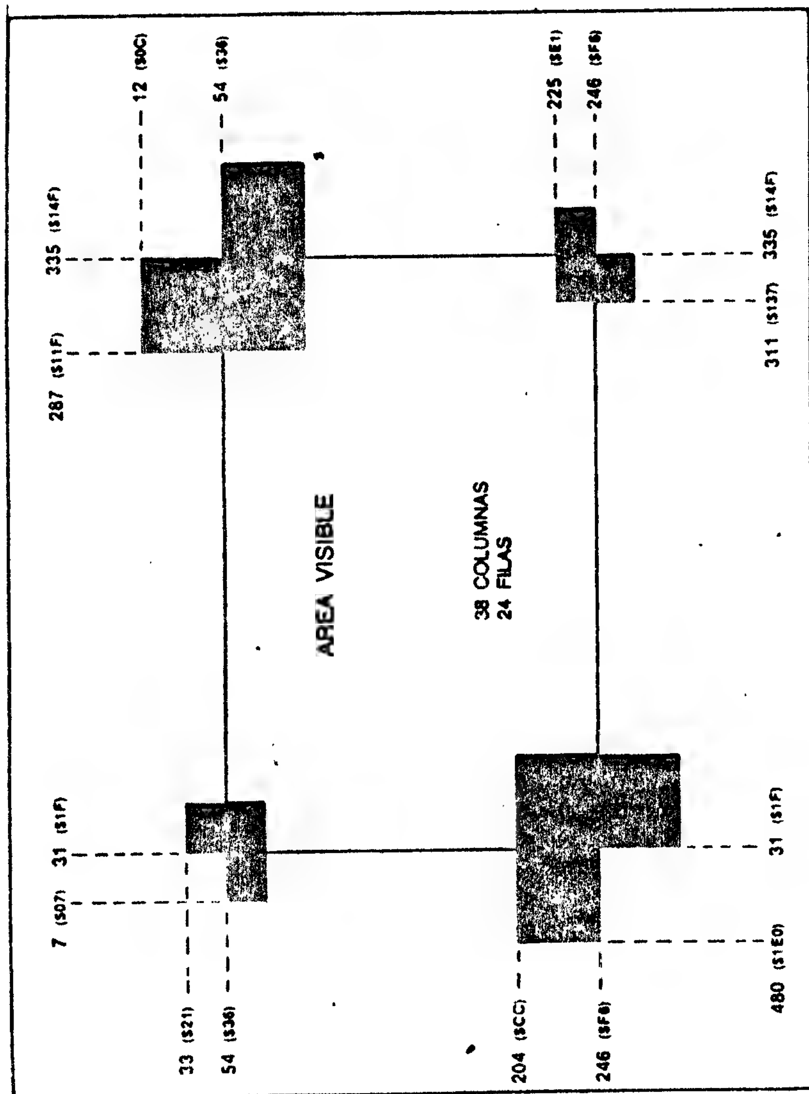


Figura 3-3. Sprite.



Posicionado de Caracteres

si la posición X del Sprite es menor que 256 (a la izquierda de la pantalla), entonces el X MSB de dicho Sprite debe ser puesto a cero. Los bits 0 a 7 del registro X MSB corresponden a los Sprites 0 a 7 respectivamente.

El siguiente programa mueve un Sprite a traves de la pantalla:

EJEMPLO:

SHIFT CLR/HOME

```
10 PRINT"J"
20 POKE2040,13
30 FORI=0TO62:POKE832+I,129:NEXT
40 V=53248
50 POKEV+21,1
60 POKEV+39,1
70 POKEV+1,100
80 FORJ=0TO347
90 HX=INT(J/256):LX=J-256*HX
100 POKEV,LX:POKEV+16,HX:NEXT
```

Cuando mueve en la dirección X Sprites expandidos que deban aparecer por la izquierda de la pantalla debe colocarlos primero fuera de la pantalla en la PARTE DERECHA de la misma. Esto ocurre porque en la parte izquierda no hay suficiente espacio para ocultar totalmente un Sprite expandido.

EJEMPLO:

SHIFT CLR/HOME

```
10 PRINT"J"
20 POKE2040,13
30 FORI=0TO62:POKE832+I,129:NEXT
40 V=53248
50 POKEV+21,1
60 POKEV+39,1:POKEV+23,1:POKEV+29,1
70 POKEV+1,100
80 J=488
90 HX=INT(J/256):LX=J-256*HX
100 POKEV,LX:POKEV+16,HX
110 J=J+1:IFJ>511THENJ=0
120 IFJ>488ORJ<348GOTO90
```

Las tablas de la figura 3–3 aclaran el posicionado de Sprites.
Usando estos valores, usted puede colocar cada Sprite en cualquier parte de la pantalla. Moviendo el Sprite un punto cada vez, es fácil conseguir un movimiento completamente uniforme.

RESUMEN DEL POSICIONADO DE SPRITES

Los Sprites no expandidos son al menos parcialmente visibles en el modo de 40 columnas por 25 lineas con los siguientes parámetros:

$$1 <= X <= 343$$
$$30 <= Y <= 249$$

En el modo de 38 columnas, cambie los parámetros de X por los siguientes:

8 <= X <= 334

En el modo de 24 lineas, cambie los parámetros de Y por los siguientes:

34 <= Y <= 245

Los Sprites expandidos son al menos parcialmente visibles en el modo de 40 columnas por 25 lineas con los siguientes parámetros:

489 >= X <= 343
9 >= Y <= 249

En el modo de 38 columnas cambie los parámetros de X por los siguientes:

496 >= X <= 334

En el modo de 24 lineas cambie los parámetros de Y por los siguientes:

13 <= Y <= 245

PRIORIDADES DE LOS SPRITES EN PANTALLA

Los Sprites tienen capacidad para cruzar por delante o detrás de los objetos situados en la pantalla. Esto le permite crear juegos con efecto tridimensional.
La prioridad entre Sprites es fija. Esto significa que el Sprite 0 tiene la más alta prioridad, le sigue el 1, y así hasta el 7, que tiene la prioridad más baja. En otras palabras, si se cruzan el Sprite 1 y el 6, el 1 pasará por encima del 6.
Planeando la misión que debe hacer cada Sprite, podrá asignarles el número que más se ajuste a su prioridad. Utilice los Sprites de número alto (5,6,7) para los objetos que deben estar en último término, y los de número bajo (0,1,2) para los que deban pasar por encima de los demás.

> **NOTA**. Es posible crear un efecto de "ventana". Si un Sprite con alta prioridad tiene zonas transparentes, al cruzar con otro de prioridad inferior este último será visible por las zonas transparentes del primero.

La prioridad del Sprite con la pantalla se fija mediante el registro PRIORIDAD SPRITE-PANTALLA, en la posición 53275 ($D01B). Cada Sprite tiene un bit en este registro. Si este bit está a cero, el Sprite tiene prioridad sobre los datos de la pantalla. Si el bit está a 1, el Sprite pasara por detrás de los datos.

DETECCION DE COLISIONES

Uno de los aspectos más interesantes del VIC-II chip es su capacidad para detectar colisiones. Estas pueden ser detectadas entre Sprites, o entre Sprites y datos. Ocurre una colision cuando una parte visible del Sprite se sobrepone a una parte visible de otro Sprite o carácter en la pantalla.

COLISION DE SPRITES ENTRE SI

Las colisiones entre Sprites son reconocidas por el ordenador en el registro del VIC-II de colisión entre Sprites, en la posición 53278 ($D01E). Cada Sprite tiene un bit en este registro. Si el bit está a 1, el Sprite en cuestión esta envuelto en una colisión. Los bits del registro quedan con el mismo valor hasta que el registro es leido (PEEK). Cuando se lee, el registro se borra automáticamente, por lo que es una buena idea guardar el contenido del registro en una variable hasta que haya realizado todas las operaciones que se deriven de una colisión.

> **NOTA**: Las colisiones se detectan incluso cuando los Sprites están fuera de la pantalla

COLISION ENTRE SPRITES Y DATOS

Las colisiones entre Sprites y datos se detectan en el registro de control del VIC-II colisión Sprite-datos, colocado en la posición 53279 ($D01F).
Cada Sprite tiene un bit en este registro. Si el bit está a 1, entonces el Sprite correspondiente está implicado en una colisión. Los bits de este registro permanecen ajustados hasta que el registro es leido. Entonces dicho registro es borrado automáticamente, por lo que es una buena idea conservar el valor del registro en una variable para posteriores cálculos.

> **NOTA**: El dato 01 en el modo MULTICOLOR se considera transparente para colisiones, aunque aparezca de otro color en la pantalla. Puede ser una buena idea para evitar confusiones que los datos 01 no esten en los bordes del Sprite en el modo multicolor

```
10 REM EJEMPLO SPRITE 1...
20 REM EL GLOBO DE AIRE CALIENTE DE NUEVO
30 VIC=13*4096:REM AQUI ES DONDE LOS REGISTROS DEL VIC EMPIEZAN
35 POKEVIC+21,1:REM FACILITA EL SPRITE 0
36 POKEVIC+33,14:REM PONE EL FONDO DE COLOR AZUL CLARO
37 POKEVIC+23,1:REM EXPANDE EL SPRITE 0 EN LA Y
38 POKEVIC+29,1:REM EXPANDE EL SPRITE 0 EN LA X
40 POKE2040,192:REM PONE EL PUNTERO DEL SPRITE 0
180 POKEVIC+0,100:REM PONE LA POSICION X DEL SPRITE 0
190 POKEVIC+1,100:REM PONE LA POSICION Y DEL SPRITE 0
220 POKEVIC+39,1:REM PONE EL COLOR DEL SPRITE 0
250 FORY=0TO63:REM CONTADOR DE BYTES CON EL BUCLE DEL SPRITE
300 READA:REM LEE EN UN BYTE
310 POKE192*64+Y,A:REM ALMACENA DATOS EN EL AREA DEL SPRITE
320 NEXTY:REM CIERRA EL BUCLE
330 DX=1:DY=1
340 X=PEEK(VIC):REM MIRA LA POSICION X DEL SPRITE 0
350 Y=PEEK(VIC+1):REM MIRA LA POSICION Y DEL SPRITE 0
360 IFY=50ORY=208THENDY=-DY:REM SI Y ESTA EN EL MARGEN DE LA
370 REM PANTALLA,ENTONCES INVIERTE DELTA Y
```

```
380 IFX=24AND(PEEK(VIC+16)AND1)=0THENDX=-DX:REM SI EL SPRITE ESTA
390 REM TOCANDO EL MARGEN IZQUIERDO,ENTONCES LO INVIERTE
400 IFX=40AND(PEEK(VIC+16)AND1)=1THENDX=-DX:SI EL SPRITE ESTA
410 REM TOCANDO EL MARGEN DERECHO,ENTONCES LO INVIERTE
420 IFX=255ANDDX=1THENX=-1:SIDE=1
430 REM VA AL OTRO LADO DE LA PANTALLA
440 IFX=0ANDDX=-1THENX=256:SIDE=0
450 REM VA AL OTRO LADO DE LA PANTALLA
460 X=X+DX:REM SUMA DELTA X A X
470 X=XAND255:REM COMPRUEBA QUE LAIX ESYA EN EL RANGO AUTORIZADO
480 Y=Y+DY:REM SUMAIDELTA Y A Y
485 POKEVIC+16,SIDE
490 POKEVIC,X:REM PONE LA NUEVA X EN LA POSICION X DEL SPRITE 0
510 POKEVIC+1,Y:REM PONE LA NUEVA Y EN LA POSICION Y DEL SPRITE 0
530 GOTO 340
600 REM####DATOS DE LOS SPRITE###
610 DATA0,127,0,1,255,192,3,255,224,3,231,224
620 DATA7,217,240,7,223,240,7,217,240,3,231,224
630 DATA3,255,224,3,255,224,2,255,160,1,127,64
640 DATA1,62,64,0,156,128,0,156,128,0,73,0,0,73,0
650 DATA0,62,0,0,62,0,0,62,0,0,28,0,0
```

```
10 REM SPRITE EJEMFLO 2
20 REM EL GLOBO DE AIRE CALIENTE OTRA VEZ
30 VIC=13*4096:REM AQUI ES DONDE EMPIEZAN LOS REGISTROS DEL VIC
35 POKEVIC+21,63:REM AUTORIZA LOS SPRITES DEL 0 AL 5
36 POKEVIC+33,14:REM PONE EL FONDO DE COLOR AZUL CLARO
37 POKEVIC+23,3:REM EXPANDE LOS SPRITES 0 Y 1 EN LA Y
38 POKEVIC+29,3:REM EXPEHDE LOS SPRITES 0 Y 1 EN LA X
40 POKE2040,192:REM COLOCA EL PUNTERO DEL SPRITE 0
50 POKE2041,193:REM COLOCA EL PUNTERO DEL SPRITE 1
60 POKE2042,192:REM COLOCA EL PUNTERO DEL SPRITE 2
70 POKE2043,193:REM COLOCA EL PUNTERO DEL SPRITE 3
80 POKE2044,192:REM COLOCA EL PUNTERO DEL SPRITE 4
90 POKE2045,193:REM COLOCA EL PUNTERO DEL SPRITE 5
100 POKEVIC+4,30:REM COLOCA LA POSICION X DEL SPRITE 2
110 POKEVIC+5,58:REM COLOCA LA POSICION Y DEL SPRITE 2
120 POKEVIC+6,65:REM COLOCA LA POSICION X DEL SPRITE 3
130 POKEVIC+7,58:REM COLOCA LA POSICION Y DEL SPRITE 3
140 POKEVIC+8,100:REM COLOCA LA POSICIONX DEL SPRITE 4
```



```
150 POKEVIC+9,58:REM COLOCA LA POSICION Y DEL SPRITE 4
160 POKEVIC+10,100:REM COLOCA LA POSICION X DEL SPRITE 5
170 POKEVIC+11,58:REM COLOCA LA POSICION Y DEL SPRITE 5
          CTRL 2
175 PRINT"♥"TAB(15)"ESTO ES DOS SPRITES DE ALTA RESOLUCION"
          SHIFT CLR/HOME
176 PRINTTAB(55)"UNO ENCIMA DE OTRO"
180 POKEVIC+10,100:REM COLOCA LA POSICION X DEL SPRITE 0
190 POKEVIC+1,100:REM COLOCA LA POSICION Y DEL SPRITE 0
200 POKEVIC+2,100:REM COLOCA LA POSICION X DEL SPRITE 1
210 POKEVIC+3,100:REM COLOCA LA POSICION Y DEL SPRITE 1
220 POKEVIC+39,1:REM COLOCA EL COLOR DEL SPRITE 0
230 POKEVIC+41,1:REM COLOCA EL COLOR DEL SPRITE 2
240 POKEVIC+43,1:REM COLOCA EL COLOR DEL SPRITE 4
250 POKEVIC+40,6:REM COLOCA EL COLOR DEL SPRITE 1
260 POKEVIC+42,6:REM COLOCA EL COLOR DEL SPRITE 3
270 POKEVIC+44,6:REM COLOCA EL COLOR DEL SPRITE 5
280 FORX=192TO193:REM EL INICIO DEL BUCLE QUE DEFINE LOS SPRITES
290 FORY=0TO63:REM CONTADOR DE BYTES CON UN BUCLE DE SPRITES
300 READA:REM LEER EN EL BYTE A
310 POKEX*64+Y,A:REM ALMACENA LOS DATOS EN EL AREA DEL SPRITE
320 NEXTY,X:REM CIERRA LOS BUCLES
330 DX=1:DY=1
340 X=PEEK(VIC):REM MIRA LA POSICION X DEL SPRITE 0
350 Y=PEEK(VIC+1):REM MIRA LA POSICION Y DEL SPRITE 0
360 IFY=50ORY=208THENDY=-DY:REM SI Y ESTA EN EL MARGEN DE
370 REM LA PANTALLA,ENTONCES INVIERTE DELTA Y
380 IFX=24AND(PEEK(VIC+16)AND1)=0THENDX=-DX:REM SI EL SPRITE ESTA
390 REM TOCANDO EL MARGEN IZQUIERDO,ENTONCES SE INVIERTE
400 IFX=40AND(PEEK(VIC+16)AND1)=1THENDX=-DX:REM SI EL SPRITE ESTA
410 REM TOCANDO EL MARGEN DERECHO,ENTONCES SE INVIERTE
420 IFX=255ANDDX=1THENX=-1:SIDE=3
430 REM VA AL OTRO LADO DE LA PANTALLA
440 IFX=0ANDDX=-1THENX=256:SIDE=0
450 REM VA AL OTRO LADO DE LA PANTALLA
460 X=X+DX:REM SUMA DELTA X A X
470 X=XAND255:REM COMPRUEBA DE QUE X TIENE EL RANGO PERMITIDO
480 Y=Y+DY:REM SUMA DELTA Y A Y
485 POKEVIC+16,SIDE
490 POKEVIC,X:REM PONE EL NUEVO VALOR DE X EN LA POSICION X DEL SPRITE 0
500 POKEVIC+2,X:REM PONE EL NUEVO VALOR DE X EN LA POSICION X DEL SPRITE 1
```

```
510 POKEVIC+1,Y:REM PONE EL NUEVO VALOR DE Y EN LA POSICION Y DEL SPRITE 0
520 POKEVIC+3,Y:REM PONE EL NUEVO VALOR DE Y EN LA POSICION Y DEL SPRITE 1
530 GOTO340
600 ####DATOS DEL SPRITE#####
610 DATA0,255,0,3,153,192,7,24,224,7,56,224,14,126,112,14,126,112,14,126,112
620 DATA6,126,96,7,56,224,7,56,224,1,56,128,0,153,0,0,90,0,0,56,0
630 DATA 0,56,0,0,0,0,0,0,0,0,126,0,0,42,0,0,84,0,0,40,0,0
640 DATA0,0,0,0,102,0,0,231,0,0,195,0,1,129,128,1,129,128,1,129,128
650 DATA1,129,128,0,195,0,0,195,0,4,195,32,2,102,64,2,36,64,1,0,128
660 DATA1,0,128,0,153,0,0,153,0,0,0,0,6,84,0,0,42,0,0,20,0,0
```

```
10 REM EJEMPLO DE SPRITES 3...
20 REM EL GORF DE AIRE CALIENTE
30 VIC=53248:REM AQUI  EMPIEZAN LOS REGISTROS DEL VIC
35 POKEVIC+21,1:REM FACILITA EL SPRITE 0
36 POKEVIC+33,14:REM PONE EL FONDO DE COLOR AZUL CLARO
37 POKEVIC+23,1:REM EXPANDE EL SPRITE 0 EN Y
38 POKEVIC+29,1:REM EXPANDE EL SPRITE 0 EN X
40 POKE2040,192:REM PONE EL POINTER DEL SPRITE 0
50 POKEVIC+28,1:REM CONECTA EL MULTICOLOR
60 POKEVIC+37,7:REM PONE EL MULTICOLOR 0
70 POKEVIC+38,4:REM PONE EL MULTICOLOR 1
180 POKEVIC+0,100:REM PONE LA POSICION X DEL SPRITE 0
190 POKEVIC+1,100:REM PONE LA POSICION Y DEL SPRITE 0
220 POKEVIC+39,2:REM COLOCA EL COLOR DEL SPRITE 0
290 FORY=0TO63:REM CONTADOR DE BYTES CON UN BUCLE DE SPRITES
300 READA:REM LEE EN EL BYTE A
310 POKE12288+Y,A:REM ALMACENA LOS DATOS EN EL AREA DE SPRITE
320 NEXTY:REM CIERRA EL BUCLE
330 DX=1:DY=1
340 X=PEEK(VIC):REM MIRA LA POSICION X DEL SPRITE 0
350 Y=PEEK(VIC+1):REM MIRA LA POSICION Y DEL SPRITE 0
360 IFY=50ORY=208THENDY=-DY:REM SI Y ESTAEN EL MARGEN DE LA
370 REM PANTALLA,ENTONCES INVIERTE DELTA Y
380 IF X=24AND(PEEK(VIC+16)AND1)=0THENDX=-DX:REM SI EL SPRITE ESTA
390 REM TOCANDO EL MARGEN IZQUIERDO,ENTONCES LO INVIERTE
400 IFX=40AND(PEEK(VIC+16)AND1)=1THENDX=-DX:REM SI EL SPRITE ESTA
410 REM TOCANDO EL MARGEN DERECHO,ENTONCES SE INVIERTE
420 IFX=255ANDDX=1THENX=-1:SIDE1
430 REM VA AL OTRO LADO DE LA PANTALLA
440 IFX=0ANDDX=-1THENX=256:SIDE=0
450 REM VA AL OTRO LADO DE LA PANTALLA
```



```
460 X=X+DX:REM SUMA DELTA X A X
470 X=XAND255:REM COMPRUEBA QUE X ESTE EN EL RANGO APROPIADO
480 Y=Y+DY:REM SUMA DELTA Y A Y
485 POKEVIC+16,SIDE
490 POKEVIC,X:REM PONE EL NUEVO VALOR DE X EN LA POSICION X DEL SPRITE 0
510 POKEVIC+1,Y:REM PONE EL NUEVO VALOR DE Y EN LA POSICION Y DEL SPRITE 0
520 GETA$:REM ESPERA QUE EL USUARIO PULSE UNA TECLA
521 IFA$="M"THENPOKEVIC+28,1:REM EL USUARIO HA ELEGIDO EL MULTICOLOR
522 IFA$="H"THENPOKEVIC+28,0:REM EL USUARIO HA ELEGIDO ALTA RESOLUCION
530 GOTO340
600 REM#####DATOS DEL SPRITE#####
610 DATA64,0,1,16,170,4,6,170,144,10,170,160,42,170,168,41,105,104,169,235,106
620 DATA169,235,106,169,235,106,170,170,170,170,170,170,170,170,170,170,170,170
630 DATA166,170,154,169,85,106,170,85,170,42,170,168.10,170,160,1,0.64,1,0,64
640 DATA5,0,80,0
```

OTRAS CARACTERISTICAS GRAFICAS

VACIADO DE PANTALLA

El bit 4 del registro de control del VIC-II controla el vaciado (desactivación) de la pantalla. Este registro se encuentra en la posición 53265 ($D011). Cuando el bit está a 1, la pantalla es normal. Si colocamos a cero el cuarto bit de este registro, la pantalla entera cambia al mismo color del borde.
El siguiente POKE desactiva la pantalla. Los datos no se pierden, únicamente no se muestran.

POKE 53265,PEEK(53265)AND 239

Para volver a pantalla normal, use el siguiente POKE:

POKE 53265,PEEK(53265)OR 16

> **NOTA:** Desactivando la pantalla, el procesador trabaja más deprisa. Esto significa que su programa será también ejecutado más rápidamente.

REGISTRO DE BARRIDO

El registro de barrido se encuentra en la posición 53266 ($D012), correspondiente al VIC-II. Es un registro con doble propósito: Si usted lo lee, obtendrá los 8 bits más bajos de la posición actual de barrido de la pantalla. El bit más significativo de este registro se encuentra en la posición 53265 ($D011). Usted puede usar el registro de barrido para que los cambios en la pantalla se produzcan sin perturbaciones en la misma. Los cambios en la pantalla se deben efectuar cuando el barrido no está

presente en el área visible de la misma, es decir, con un valor entre 51 y 251. Cuando este registro es escrito (incluyendo el MSB), el valor asignado se guarda para usarlo en la comparación de barrido. Cuando el valor actual de barrido es el igual al valor escrito anteriormente, un bit en el registro de interrupción del VIC-II en la posición 53273 ($D019) es puesto a 1.

NOTA: Si el propio bit es activado (colocado a 1), ocurre una interrupción (IRQ).

REGISTRO DE ESTADO DE INTERRUPCION

Este registro muestra el estado actual de cualquier causa de interrupción. El bit 2 de este registro se coloca a 1 cuando dos Sprites chocan . Lo mismo es cierto, en la correspondiente relación 1 a 1, para los bits 0 a 3, como se muestra en la tabla siguiente. El bit 7 se coloca a 1 cuando ocurre una interrupción.
El registro de estado de interrupción se encuentra situado en la posición 53273 ($D019), y es como sigue:

CLAVE	BIT#	DESCRIPCION
IRST	0	Activado cuando el contador de barrido es igual al número almacenado
IMDC	1	Activado por colisión Sprite-Dato. (Sólo la primera vez, hasta que es reinicializado)
IMMC	2	Activado por colisión Sprite-Sprite. (Sólo la primera vez, hasta que es reinicializado)
ILP	3	Activado por transición negativa del lápiz óptico. (1 por cuadro)
IRQ	7	Activado por interrupciones

Cuando un bit de interrupción ha sido activado, es "cerrado" y debe ser limpiado escribiendo un 1 en este bit cuando esté listo para manejarlo. Esto le permite el manejo de interrupciones selectivas, sin tener que almacenar los otros bits de interrupción.
El **REGISTRO DE INTERRUPCIONES** está situado en la posición 53274 ($D01A). Tiene el mismo formato que el registro de estado de interrupción. A menos que el correspondiente bit del registro de interrupciones este a 1, ninguna interrupción del origen correspondiente puede ocurrir. Este registro puede ser leido para información, pero no se pueden generar interrupciones.
Para pedir información sobre una interrupción, el correspondiente bit de este registro (como se muestra en la tabla de arriba) debe ponerse a 1.
Este potente registro le permite dividir en zonas la pantalla. Usted puede tener media pantalla en modo "Bit Map", la otra mitad en modo texto, más de 8 Sprites a la vez, etc. El secreto es usar las interrupciones adecuadamente. Por ejemplo, si usted desea la mitad superior de la pantalla en modo Bit Map y la inferior en modo texto, coloque el registro de barrido (tal como se explicó antes) en la mitad de la pantalla. Cuando aparece la interrupción, el VIC-II toma los caracteres de la ROM. Entonces, coloque el registro de barrido al principio de la pantalla. Cuando ocurre la interrupción en la parte superior de la pantalla, el VIC-II toma los caracteres de la RAM (Bit Map).

Usted puede mostrar más de ocho Sprites a la vez del mismo modo. Desafortunadamente, el BASIC no es lo bastante rápido para que esta técnica funcione bien. Si desea usar las interrupciones de pantalla, deberá trabajar en lenguaje máquina.

COMBINACIONES SUGERIDAS DE COLOR DE PANTALLA Y CARACTERES

El color en un aparato de televisión tiene un límite en la habilidad para colocar ciertos colores al lado de otros en la misma línea. Algunas combinaciones de pantalla y caracteres producen imágenes deficientes. La siguiente tabla le muestra las combinaciones posibles, y de entre ellas las más adecuadas para trabajar.

COLOR DEL CARACTER

COLOR DE LA PANTALLA	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	●	X	●	●	■	X	●	●	X	●	●	●	●	●	●
1.	●	X	●	X	●	●	●	X	■	●	■	●	●	X	●	●
2	X	●	X	X	■	X	X	●	●	X	●	X	X	X	X	■
3	●	X	X	X	X	■	●	X	X	X	X	■	X	X	■	X
4	●	■	X	X	X	X	X	X	X	X	X	X	X	X	X	■
5	●	■	X	■	X	X	X	X	X	X	X	■	X	●	X	■
6	■	●	X	●	X	X	X	X	X	X	X	X	X	■	●	●
7	●	X	●	X	X	X	■	X	■	●	■	●	●	X	X	X
8	■	●	●	X	X	X	X	●	X	●	X	X	X	X	X	■
9	X	●	X	X	X	X	X	●	●	X	●	X	X	X	X	●
10	■	■	●	X	X	X	X	■	X	●	X	X	X	X	X	■
11	●	●	X	■	X	X	X	●	X	X	X	X	●	●	■	●
12	●	●	■	X	X	X	■	X	X	■	X	●	X	X	X	●
13	●	X	X	X	X	●	■	X	X	X	X	●	X	X	X	X
14	●	●	X	●	X	X	●	X	X	X	X	■	X	X	X	■
15	●	●	●	X	■	■	●	X	X	■	■	●	●	X	■	X

■ = EXCELENTE
● = BUENA
X = REGULAR

PROGRAMACION DE SPRITES. REVISION

Por si usted ha tenido problemas con los gráficos, esta sección ha sido preparada como una aproximación elemental a la confección de Sprites.

CONFECCION DE SPRITES EN BASIC–UN PROGRAMA CORTO

Hay como mínimo tres técnicas distintas de programación en BASIC que le permiten crear imágenes gráficas y dibujos animados en su Commodore 64. Usted puede usar el juego de caracteres gráficos incorporados (vea Apéndice B). Puede programar sus propios caracteres (vea Pág. 86). O...lo mejor de todo...puede usar los "Gráficos Sprites" incorporados a su ordenador. Para ilustrarle cuan fácil es, he aquí uno de los más cortos programas en BASIC que le permite crear un Sprite:

SHIFT CLR/HOME

```
10 PRINT"J"
20 POKE2040,13
30 FORS=832TO832+62:POKES,255:NEXT
40 V=53248
50 POKEV+21,1
60 POKEV+39,1
70 POKEV,24
80 POKEV+1,100
```

Este programa incluye los ingredientes "clave" que necesita para crear cualquier Sprite. Los números POKE se toman de la TABLA DE CONFECCION DE SPRITES de la página 176. Este programa define el primer Sprite –Sprite 0– como un rectángulo blanco en la pantalla. A continuación se explica el programa línea a línea:

LINEA 10 Limpia la pantalla.

LINEA 20 Coloca el "puntero de Sprite" para indicar al Commodore 64 de donde debe tomar la información sobre la forma del Sprite. El Sprite 0 utiliza la posición 2040, el Sprite 1 la 2041, el Sprite 2 la 2042, y así hasta el Sprite 7, que ocupa la posición 2047. Puede colocar todos los punteros de Sprite a 13 usando la siguiente línea en lugar de la 20:

FOR SP=2040 TO 2047:POKE SP,13:NEXT SP

LINEA 30 Coloca el primer Sprite (Sprite 0) en los 63 bytes de memoria RAM que empiezan en la posición 832 (cada Sprite necesita 63 bytes de memoria). El primer Sprite (Sprite 0) "reside" en las posiciones de memoria 832-894.

LINEA 40 Asigna el valor 53248 a la variable "V". Esta posición es la de inicio del VIC-II chip. De esta forma, todos los registros se usarán mediante la fórmula (V+número). El uso de esta fórmula (V+número) al hacer POKES para inicializar los

Sprites le permite ahorrar memoria y trabajar con números más pequeños. Por ejemplo, en la línea 50 se ha escrito POKE V+21. Esto es lo mismo que escribir POKE 53248+21 o POKE 53269...pero V+21 requiere menos espacio que 53269, y es más fácil de recordar.

LINEA 50 "activa" el Sprite 0. Hay 8 Sprites, numerados de 0 a 7. Para activar un Sprite individual, o una combinación de Sprites, todo lo que debe hacer es POKE V+21 seguido de un número de 0 (todos los Sprites desactivados) a 255 (todos los Sprites activados). Puede activar uno o más Sprites haciendo un POKE con los siguientes números:

TODOS	SPRITE0	SPRITE1	SPRITE2	SPRITE3	SPRITE4	SPRITE5	SPRITE6	SPRITE7	NINGUNO
V+21,255	V+21,1	V+21,2	V+21,4	V+21,8	V+21,16	V+21,32	V+21,64	V+21,128	V+21,0

POKE V+21,1 activa el Sprite 0. POKE V+21,128 activa el Sprite 7. Usted puede también activar combinaciones de Sprites. Por ejemplo, POKE V+21,129 activa los Sprites 0 y 7, gracias a la adición de los respectivos números de activación. (128+1=129) (vea la TABLA DE CONFECCION DE SPRITES en la página 176).

LINEA 60 Coloca el color del Sprite 0. Hay 16 posibles colores para cada Sprite, numerados de 0 (negro) a 15 (gris). Cada Sprite precisa un POKE distinto para definir su color, desde V+39 a V+46. POKE V+39,1 asigna el color blanco al Sprite 0. POKE V+46,15 asigna el color gris al Sprite 7. (Vea la TABLA DE CONFECCION DE SPRITES PARA MAS INFORMACION).

Cuando crea un Sprite, como se ha explicado, el Sprite ESTA EN MEMORIA hasta que lo desactiva, lo redefine o desconecta su ordenador. Esto le permite cambiar la posición, el color o incluso la forma del Sprite en modo DIRECTO o INMEDIATO, lo que es útil para propósitos de edición. Como ejemplo, ejecute el programa anterior, después escriba en modo DIRECTO (sin número de línea) y pulse la tecla RETURN:

POKE V+39,8

El Sprite de la pantalla es ahora NARANJA. Pruebe con POKE de otros valores entre 0 y 15 para ver los otros colores. Puesto que estas modificaciones las ha hecho en modo directo, al ejecutar otra vez el programa, el Sprite volverá a su color original (blanco).

LINEA 70 Determina la posición horizontal o posicion "X" del Sprite en la pantalla. Este número representa la posición de la esquina SUPERIOR IZQUIERDA del Sprite. La primera posición horizontal (X) en la que puede ver el Sprite en su televisión es 24, aunque usted puede mover el Sprite fuera de la pantalla variando este número hasta llegar a 0.

LINEA 80 Determina la posición vertical o posición "Y" del Sprite. En este programa, el Sprite esta colocado en la posición 24 horizontal (X), y en la posición 100

vertical (Y). Para probar otra posición, escriba lo siguiente en MODO DIRECTO y pulse RETURN:

POKE V,24:POKE V+1,50

Esto coloca al Sprite en la esquina superior izquierda de la pantalla. Para mover el Sprite a la esquina inferior izquierda escriba lo siguiente:

POKE V,24:POKE V+1,229

Cada número contenido en las posiciones de memoria de 832 a 895 representa un bloque de 8 puntos (pixels), con tres bloques de 8 puntos por cada línea horizontal del Sprite. El bucle de la línea 80 le dice al ordenador que haga POKE 832,255, lo que hace que los primeros 8 puntos estén todos iluminados...entonces POKE 833,255 realiza lo mismo con el siguiente bloque de 8 puntos, y así hasta llegar a la posición 894, que contiene el último grupo de 8 puntos, en la esquina inferior derecha del Sprite. Para ver mejor cómo trabaja esto, pruebe a escribir lo siguiente en MODO DIRECTO, y vea cómo el segundo grupo de ocho puntos es borrado: POKE 833,0 (para volver a la normalidad teclee POKE 833,255 o ejecute (RUN) el programa).
La siguiente línea, que usted puede añadir al programa, borra los bloques del MEDIO del Sprite creado:

```
90 FOR A=836 TO 891 STEP 3:POKE A,0:NEXT A
```

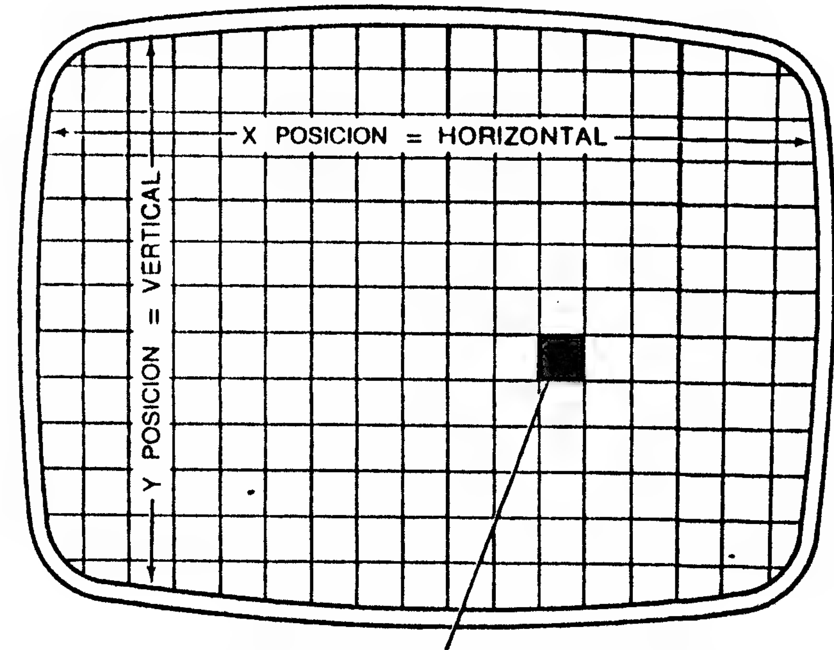
Recuerde, los puntos que forman un Sprite están agrupados en bloques de ocho. Esta línea borra el quinto bloque de 8 puntos (bloque 836), y hace lo mismo cada tres bloques hasta llegar al número 890. Pruebe POKEs con otros números desde 832 hasta 894, colocando en los que desee 255, para iluminar todos los puntos, o cero, para apagarlos.

COMPRIMA SUS PROGRAMAS DE SPRITES

He aquí una útil nota sobre compresión de programas: El programa descrito antes es realmente corto, pero aún puede serlo más utilizando técnicas de compresión. En nuestro ejemplo hemos colocado las diversas instrucciones para crear y posicionar un Sprite en líneas separadas, para que viese mejor cómo funciona el programa. Con la práctica actual, un buen programador deseará probablemente escribir el programa en SOLO DOS LINEAS ..comprimido como se muestra:

```
10PRINTCHR$(147):V=53248:POKEV+21,1:POKE2040,13:POKEV+39,1
20FORS=832TO894:POKES,255:NEXT:POKEV,24:POKEV+1,100
```

Para más "trucos" de compresión que le permitan ahorrar memoria y ejecutar el programa más rápidamente, consulte la "guía de compresión" en la Pág. 19.



Un Sprite colocado aquí debe tener definidas ambas coordenadas: posición horizontal (X) y posición vertical (Y), para que pueda ser mostrado en la pantalla.

Figura 3-4. La pantalla de T.V. se divide en una reja de coordenadas X e Y.

POSICIONADO DE SPRITES EN LA PANTALLA

La pantalla entera es dividida en una red de coordenadas X e Y, igual que un gráfico. La COORDENADA X es la posición HORIZONTAL, a lo largo de la pantalla, y la COORDENADA Y es la posición VERTICAL, arriba y abajo. (Vea Figura 3-4). Para posicionar un Sprite en la pantalla, debe hacer un POKE de DOS posiciones la posición X y la posición Y– para decir al ordenador dónde debe emplazarse la ESQUINA SUPERIOR IZQUIERDA del Sprite. Recuerde que un Sprite está compuesto de 504 puntos individuales, 24 de ancho y 21 de alto, por lo que en el caso de posicionarlo en la esquina superior izquierda de la pantalla, el Sprite será mostrado como una imagen gráfica de 24 puntos de largo y 21 de alto, iniciados en la posición X-Y que haya definido. El Sprite debe ser mostrado basándose en la esquina superior Izquierda del Sprite entero, aunque sólo una parte de el sea utilizada. Para que comprenda cómo trabaja el posicionado de X-Y, estudie el siguiente diagrama (Figura 3-5), que le muestra las posiciones de X e Y en relación con la

pantalla. Advierta que la parte GRIS del diagrama representa la parte visible de la pantalla...la parte blanca representa posiciones de FUERA de la pantalla...

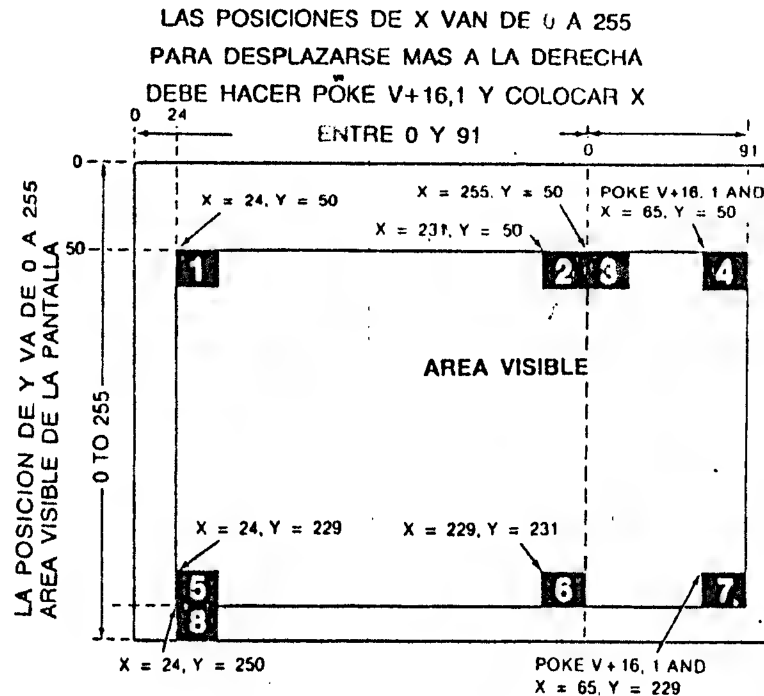


Figura 3–5. Determinando las posiciones X-Y de un Sprite.

Para mostrar un Sprite en la posición deseada, debe hacer un POKE de X e Y para cada Sprite...recordando que cada Sprite tiene su propia posición para hacer el POKE de X e Y. Las posiciones de estos POKES para cada Sprite se muestran aquí:

POKE EN LOS SIGUIENTES VALORES PARA POSICIONAR SPRITES

	SPRITE0	SPRITE1	SPRITE2	SPRITE3	SPRITE4	SPRITE5	SPRITE6	SPRITE7
SET X	V,X	V+2,X	V+4,X	V+6,X	V+8,X	V+10,X	V+12,X	V+14,X
SET Y	V+1,Y	V+3,Y	V+5,Y	V+7,Y	V+9,Y	V+11,Y	V+13,Y	V+15,Y
RIGHTX	V+16,1	V+16,2	V+16,4	V+16,8	V+16,16	V+16,32	V+16,64	V+16,128

POKE DE LA POSICION X: Los valores posibles de X van de 0 a 255, de izquierda a derecha. Los valores de 0 a 23 colocan parte del Sprite FUERA DEL AREA DE VISION en la parte izquierda de la pantalla...los valores de 24 a 255 colocan el Sprite

en el AREA DE VISION hasta la posición 255 (vea el siguiente párrafo para posiciones superiores a 255). Para colocar el Sprite en una de estas posiciones, escriba el POKE DE LA POSICION X para el Sprite que esté usando. Por ejemplo, para POKE Sprite 1 en la posición Izquierda de la pantalla escriba POKE V+2,24

VALORES DE X MAYORES DE 255: Para acceder a posiciones mayores de 255 en la pantalla, necesita un SEGUNDO POKE usando los números de "X derecha" mostrados en la tabla (Vea Figura 3-5). Normalmente, la posición de X debería pasar de 255 a 256,257,etc., pero como que los registros sólo tienen 8 bits, se debe usar un "segundo registro" para acceder a la parte derecha de la pantalla e iniciar de nuevo el valor de X en 0. Para acceder a la parte derecha de la pantalla, debe hacer POKE V+16, y un número, dependiendo del Sprite usado. Esto le da 65 posiciones adicionales (renumeradas de 0 a 65) en la parte derecha de la pantalla. (Usted puede hacer un POKE con un valor más alto de 65, como 255, con lo que el Sprite desaparecerá por la parte derecha de la pantalla.)

POKE DE LA POSICION Y: Los valores posibles de Y van de 0 a 255, contando de arriba a abajo. Los valores de 0 a 49 colocan parte del Sprite FUERA del area de visión por la parte de arriba de la pantalla. Los valores de 50 a 229 colocan al Sprite DENTRO del area de visión. Los valores de 230 a 255 colocan parte del Sprite FUERA del area de visión por la parte inferior de la pantalla.
Vea cómo trabaja el posicionado de X e Y, usando el Sprite 1. Escriba este programa:

SHIFT CLR/HOME

```
10 PRINT"♥":V=53248:POKEV+21,2:POKE2041,13:FORS=832TO895:POKES,255:NEXT
20 POKEV+40,7
30 POKEV+2,24
40 POKEV+3,50
```

Este simple programa define el Sprite 1 como un sólido rectángulo colocado en la parte superior izquierda de la pantalla. Ahora cambie la línea 40 por la que sigue:

```
40 POKE V+3,229
```

Esto mueve el Sprite a la parte inferior izquierda de la pantalla. Ahora pruebe el LIMITE DERECHO DE X del Sprite. Cambie la línea 30 como se muestra:

```
30 POKE V+2,255
```

Esto mueve el Sprite a la derecha, situándolo en el límite derecho de X. En este punto, el "bit más significativo" en el registro 16 debe ser PUESTO A 1. En otras palabras, usted debe escribir POKE V+16, y el numero mostrado en la columna de "X Der" de l tabla anterior, y colocar el registro de posición X otra vez a 0 para obtener la posición 256. Cambie la línea 30 como sigue:

```
30 POKE V+16,PEEK (V+16)OR 2: POKE V+2,0
```

POKE V+16,2 activa el bit más significativo de la posición X para el Sprite 1, colocando el mismo en la posición 256 de la pantalla. POKE V+2,0 coloca el Sprite en la NUEVA POSICION 0, que es igual a la 256.
Para que el Sprite regrese a la zona izquierda, debe colocar a cero el bit más significativo de la posición X escribiendo (para el Sprite 1):

```
POKE V+16,PEEK(V+16)AND 253
```

RESUMIENDO como trabaja el posicionado X...POKE la POSICION X para cualquier Sprite con un número entre 0 y 255.
Para acceder a una posición superior a 255, debe usar un POKE adicional (POKE V+16) que activa al bit más significativo de la posición X e inicia de nuevo la cuenta desde 0 a partir de la posición 256 de la pantalla.
Este POKE empieza de nuevo a cero a partir de la posición 256 (Ejemplo: POKEV+16, PEEK(V+16)OR 1 y POKE V,1 colocan el Sprite 0 en la posición X 257.) Para volver a la parte izquierda de la pantalla debe DESACTIVAR el bit más significativo escribiendo POKE V+16, PEEK(V+16)AND 254.

POSICIONADO DE VARIOS SPRITES EN LA PANTALLA

He aquí un programa que define TRES SPRITES DISTINTOS (0,1, y 2) en distintos colores, y los coloca en distintas partes de la pantalla:

SHIFT CLR/HOME

```
10 PRINT"J":V=53248:FORS=832TO895:POKES,255:NEXT
20 FORM=2040TO2042:POKEM,13:NEXT
30 POKEV+21,7
40 POKEV+39,1:POKEV+40,7:POKEV+41,8
50 POKEV,24:POKEV+1,50
60 POKEV+2,12:POKEV+3,229
70 POKEV+4,255:POKEV+5,50
```

Por conveniencia, los 3 Sprites han sido definidos como rectángulos, colocando la definición en el mismo sitio. La parte importante del programa es cómo se posicionan los tres Sprites. El Sprite 0 blanco está colocado en la esquina superior izquierda de la pantalla. El Sprite 1 amarillo está en la esquina inferior izquierda de la pantalla pero LA MITAD del mismo está FUERA DE LA PANTALLA. (Recuerde, 24 es la posición más pequeña de X en la que un Sprite se ve completo... y un valor inferior a 24 coloca parte del Sprite fuera de la pantalla, y usando el valor 12 como en el programa, la mitad de él está fuera de la pantalla). Finalmente, el Sprite 2 naranja, está en el LIMITE DERECHO de X (posición 255)... pero, ¿cómo puede colocar un Sprite en el área derecha de X, más a la derecha de la posición 255?

MOSTRAR UN SPRITE EN UNA POSICION SUPERIOR A LA 255

Mostrar un Sprite en una posición mayor que 255 requiere un POKE especial que coloca a 1 el bit más significativo de la posición X e inicia de nuevo las posiciones a partir de la 256. Vea cómo trabaja...

Primero, haga un POKE V+16, con el número adecuado para el Sprite con el que trabaja (compruebe la línea X Der. en la tabla X-Y... mejor use el Sprite 0). Ahora asigne una posición de X, acuérdese que el contador empieza otra vez de 0 a 256. Cambie la línea 50 por la que sigue:

```
50 POKE V+16,1:POKE V,24:POKE V+1,75
```

Esta línea hace un POKE V+16 con el número requerido para "abrir" la zona derecha de la pantalla... la nueva posición de X 24 para el Sprite 0 representa 24 puntos a la derecha de la posición 255. Para comprobar el borde derecho de la pantalla, cambie la línea 60 por:

```
60 POKE V+16,1:POKE V,65:POKE V+1,75
```

Alguna experimentación con la tabla de posicionado de Sprites le permitirá colocar los Sprites donde los precise y moverlos por la pantalla. La sección "movimiento de los Sprites" incrementará su comprensión del modo en que trabaja el posicionado de Sprites.

PRIORIDADES DE LOS SPRITES

Usted puede lograr que los Sprites se muevan POR DELANTE o por DETRAS de los demás en la pantalla. Esta increíble ilusión tridimensional está lograda gracias a las PRIORIDADES DE SPRITES incorporadas a su ordenador, que determinan que Sprites tienen prioridad sobre otros cuando 2 o más Sprites se cruzan en la pantalla.
La regla es muy sencilla: los Sprites con el número más bajo tienen prioridad sobre los de número mayor. Por ejemplo, si coloca los Sprites 0 y 1 de modo que se superpongan en la pantalla, el Sprite 0 aparecerá POR DELANTE del Sprite 1. El Sprite 0 tiene PRIORIDAD ABSOLUTA sobre los demás porque tiene el número más bajo de todos. En comparación, el Sprite 1 tiene prioridad sobre los Sprites 2–7; el Sprite 2 tiene prioridad sobre los Sprites 3–7, etc. El Sprite 7 (el último Sprite) tiene MENOS PRIORIDAD que cualquier otro Sprite, por lo que siempre pasará por detrás de los otros Sprites al cruzarse.
Para ilustrar cómo trabajan las prioridades, cambie las líneas 50,60, y 70 del programa anterior como sigue:

SHIFT CLR/HOME

```
10 PRINT"J":V=53248:FORS=832TO895:POKES,255:NEXT
20 FORM=2040TO2042:POKEM,13:NEXT
30 POKEV+21,7
40 POKEV+39,1:POKEV+40,7:POKEV+41,8
50 POKEV,24:POKEV+1,50:POKEV+16,0
60 POKEV+2,34:POKEV+3,60
70 POKEV+4,44:POKEV+5,70
```

Usted debe ver un Sprite blanco encima de uno amarillo que a su vez está encima de uno naranja. Por supuesto, ahora que conoce las prioridades de los Sprites, podrá MOVER SPRITES teniendo en cuenta su prioridad para realizar efectos de animación realmente buenos.

DIBUJAR UN SPRITE

Dibujar un Sprite Commodore es igual que colorear los espacios vacíos de un libro para colorear. Cada Sprite consiste en una serie de pequeños puntos llamados pixels. Para dibujar un Sprite, lo único que debe hacer es "colorear" algunos de los puntos.

Vea la malla de confección de Sprites en la figura 3-6. Es parecida a lo que sería un Sprite totalmente en blanco:

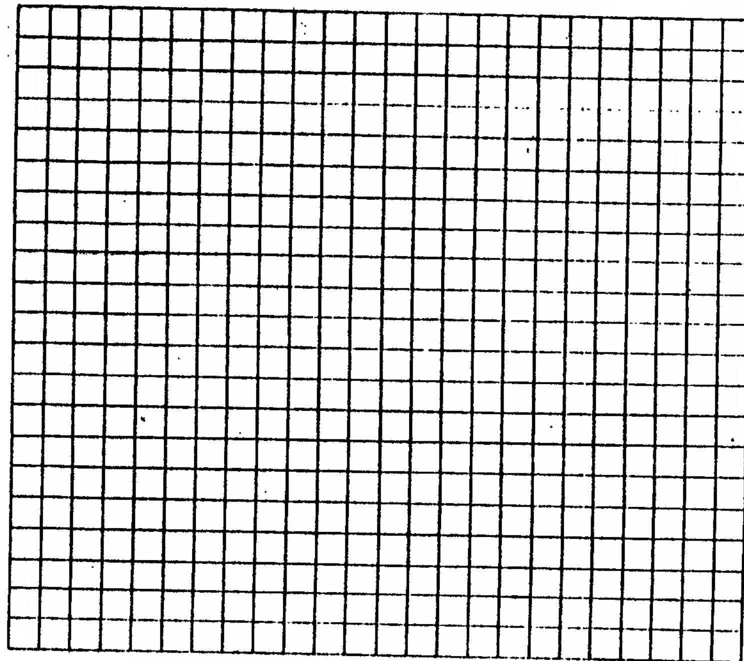


Figura 3–6. Malla de confección de Sprites.

Cada pequeño cuadro representa un punto en su Sprite. Hay 24 puntos a lo largo y 21 a lo alto, lo que hace un total de 504 puntos para todo el Sprite. Para que el Sprite represente alguna cosa, usted debe colorear los puntos que desee usando un PROGRAMA especial... pero, ¿cómo puede controlar 504 puntos individuales? Ahí es donde la programacion le ayuda a usted. En lugar de escribir 504 números separados, sólo tendrá que escribir 63 números para cada Sprite. Veamos cómo se logra...

CREACION DE UN SPRITE...PASO A PASO

Para hacer esto de la forma más fácil para usted, hemos realizado esta simple guía que le explica cómo crear un Sprite paso a paso, con la que esperamos ayudarle a confeccionar fácil y rápidamente sus propios Sprites.

PASO 1:

Escriba el programa siguiente para confeccionar Sprites EN UNA HOJA DE PAPEL... advierta cómo la línea 100 y siguientes contienen una sección DATA especial que contiene 63 números para crear un Sprite.

SHIFT CLR/HOME

```
10 PRINT"♥":POKE53280,5:POKE53281,6
20 V=53248:POKEV+34,3
30 POKE53269,4:POKE2042,13
40 FORN=0TO62:READQ:POKE832+N,Q:NEXT

100 DATA255,255,255
101 DATA128,0,1
102 DATA128,0,1
103 DATA128,0,1
104 DATA144,0,1
105 DATA144,0,1
106 DATA144,0,1
107 DATA144,0,1
108 DATA144,0,1
109 DATA144,0,1
110 DATA144,0,1
111 DATA144,0,1
112 DATA144,0,1
113 DATA144,0,1
114 DATA128,0,1
115 DATA128,0,1
116 DATA128,0,1
117 DATA128,0,1
118 DATA128,0,1
119 DATA128,0,1
120 DATA255,255,255
200 X=200:Y=100:POKE53252,X:POKE53253,Y
```

PASO 2:

Coloree los puntos deseados en la tabla para confección de Sprites de la página 162 (o use una hoja de papel para gráficos... recuerde: un Sprite tiene 24 puntos de ancho y 21 de alto). Le sugerimos que utilice un lápiz y dibuje claro para que pueda distinguir la malla. Puede crear la imagen que desee, pero para nuestro ejemplo dibujamos un simple cuadro.

PASO 3:

Mire los primeros OCHO puntos. Cada columna de puntos tiene un número (128,64, 32,16,8,4,2,1). Este especial tipo de suma forma parte de la ARITMETICA BINARIA que es usada por la mayoría de ordenadores como una forma especial de contar. Esta es una imagen ampliada de los primeros ocho puntos (o pixels) en la parte superior izquierda del Sprite:

128	64	32	16	8	4	2	1

PASO 4:

Sume los números de los puntos COLOREADOS. Este primer grupo de puntos está completamente coloreado, por lo que la suma da como resultado 255.

PASO 5:

Entre este número como el primero de la instrucción DATA en la línea 100 del programa para confeccionar Sprites que se mostró anteriormente. Entre 255 para el segundo y tercer bloque de 8 puntos.

PASO 6:

Mire los OCHO PRIMEROS PUNTOS EN LA SEGUNDA FILA del Sprite. Sume los valores de los puntos coloreados. Puesto que sólo hay un punto coloreado, el valor total es 128. Entre este número en la instrucción DATA de la línea 101.

128	64	32	16	8	4	2	1
▒							

PASO 7:

Sume los valores del siguiente grupo de 8 puntos (que es cero porque no hay ningún punto coloreado) y éntrelo en la línea 101. Pase ahora al siguiente grupo y repita el proceso por cada GRUPO DE OCHO PUNTOS (hay 3 grupos por cada línea, y un total de 21 líneas). Esto le da un total de 63 números. Cada número representa UN grupo de 8 puntos, y los 63 grupos de 8 equivalen a 504 puntos individuales. Tal vez la mejor forma de comprender el programa sea la siguiente... cada línea del programa representa UNA LINEA del Sprite. Cada uno de los 3 números en una línea representan UN GRUPO DE OCHO PUNTOS. Y cada número le indica al ordenador los puntos que están coloreados y los que no.

PASO 8:

COMPRIMA SU PROGRAMA COLOCANDO JUNTOS TODOS LOS NUMEROS QUE COMPONEN LAS INSTRUCCIONES DATA, COMO SE MUESTRA EN EL PROGRAMA DE ABAJO. Advierta por que le hemos pedido que escribiese el programa en una hoja de papel. Hemos dicho esto por una buena razón: Las líneas de instrucciones DATA 100-120 en el programa del PASO 1 sirven sólo para ayudarle a comprender la relación entre los números y los grupos de puntos de su Sprite. El programa final debe quedar "comprimido" de la siguiente forma:

```
10 PRINT"J":POKE53280,5:POKE53281,6
20 V=53248:POKEV+34,3
30 POKE53269,4:POKE2042,13
40 FORN=0TO62:READQ:POKE832+N,Q:NEXT
100 DATA255,255,255,128,0,1,128,0,1,128,0,1,144,0,1,144,0,1,144,0,1,144,0,1
101 DATA144,0,1,144,0,1,144,0,1,144,0,1,144,0,1,144,0,1,128,0,1,128,0,1
102 DATA128,0,1,128,0,1,128,0,1,128,0,1,255,255,255
200 X=200:Y=100:POKE53252,X:POKE53253,Y
```


MUEVA SU SPRITE POR LA PANTALLA

Ahora que usted ha creado un Sprite, puede hacer cosas muy interesantes con él. Para mover uniformemente su Sprite por la pantalla, incluya estas dos líneas en su programa:

```
50 POKE V+5,100:FOR X=24 TO 255:POKE V+4,X:NEXT:POKE V+16,4
55 FOR X=0 TO 65:POKE V+4,X:NEXT:POKE V+16,0:GOTO 50
```

LINEA 50: hace un POKE de la posición Y a 100 (pruebe 50 o 229 para variar). Entonces inicia un bucle FOR...NEXT con POKEs de la posición X desde 0 a 255, en orden. Cuando llega a la posición 255, se hace un POKE de la posición X derecha (POKE V+16,2), que es preciso para acceder a la zona derecha de la pantalla.
LINEA 55: contiene un bucle FOR...NEXT que continúa haciendo POKEs de la posición X en las últimas 65 posiciones de la pantalla. Advierta que el valor de X ha sido puesto a 0 porque usa el registro de X DERECHA (POKE V+16,2) para continuar por la parte derecha de la pantalla.
Esta línea finaliza indicando un regreso a la línea 50 (GOTO 50). Si desea que su Sprite SOLO pase una vez por la pantalla y desaparezca, elimine el GOTO 50 de la línea 55.
He aquí unas líneas que mueven su Sprite a derecha y a izquierda:

```
50 POKE V+5,100:FOR X=24 TO 255:POKE V+4,X:NEXT:POKE
V+16,4:FOR X=0 TO 65:POKE V+4,X:NEXT X
55 FOR X=65 TO 0 STEP-1:POKE V+4,X:NEXT:POKE V+16,0:FOR X=255
TO 24 STEP-1: POKE V+4,X:NEXT
60 GOTO 50
```

¿Ve usted cómo trabaja este programa? Este programa es el mismo que el previo, excepto que al encontrar el borde derecho de la pantalla, DA MARCHA ATRAS y vuelve en la misma dirección. Esto es lo que hace el STEP–1... indica al programa que haga POKE del valor de X de 65 a 0 en la parte derecha de la pantalla, y entonces de 255 a 0 en la parte izquierda de la pantalla, pasando una posición –1 cada vez.

SCROLL VERTICAL

Este tipo de movimiento de Sprite se llama "Scroll". Para que el Sprite se mueva en la dirección Y (vertical), sólo debe usar UNA LINEA. BORRE LAS LINEAS 50 y 55 escribiendo el número de línea de ambas y pulsando RETURN, de la siguiente forma:

50 (RETURN)
55 (RETURN)

Ahora entre la línea 50 otra vez como sigue:

```
50 POKE V+4,24:FOR Y=0 TO 255:POKE V+5,Y:NEXT
```

EL RATON BAILARIN-UN EJEMPLO DE PROGRAMA CON SPRITES

Algunas veces las técnicas descritas en este manual son difíciles de comprender, por lo que hemos creado el programa llamado "El Ratón Bailarín de Miguel." Este

programa usa tres distintos Sprites en una graciosa animación con efectos de sonido, y –para ayudarle a comprender cómo funciona– hemos incluido la explicación de CADA COMANDO para que vea exactamente cómo está construido el programa:

```
5 S=54272:POKES+24,15:POKES,220:POKES+1,68:POKES+5,15:POKES+6,215
10 POKES+7,120:POKES+8,100:POKES+12,15:POKES+13,215
              SHIFT CLR/HOME
15 PRINT"J":V=53248:POKEV+21,1
20 FORS1=12288TO12350:READQ1:POKES1,Q1:NEXT
25 FORS2=12352TO12414:READQ2:POKES2,Q2:NEXT
30 FORS3=12416TO12478:READQ3:POKES3,Q3:NEXT
35 POKEV+39,15:[illegible],?8
                 CTRL 2                  [illegible]
40 PRINTTAB(160)"SOY EL RATON BAILARIN!J"
45 P=192
50 FORX=0TO347STEP3
55 RX=INT(X/256):LX=X-RX*256
60 POKEV,LX:POKEV+16,RX
70 IFP=192THENGOSUB200
75 IFP=193THENGOSUB300
80 POKE2040,P:FORT=1TO60:NEXT
85 P=P+1:IFP>194THENP=192
90 NEXT
95 END
100 DATA30,0,120,63,0,252,127,129,254,127,129,254,127,189,254,127,255,254
101 DATA63,255,252,31,187,248,3,187,192,1,255,128,3,189,192,1,231,128,1,255,0
102 DATA31,255,0,0,124,0,0,254,0,1,199,32,3,131,224,7,1,192,1,192,0,3,192,0
103 DATA30,0,120,63,0,252,127,129,254,127,129,254,127,189,254,127,255,254
104 DATA63,255,252,31,221,248,3,221,192,1,255,128,3,255,192,1,195,128,1,231,3
105 DATA31,255,255,0,124,0,0,254,0,1,199,0,7,1,128,7,0,204,1,128,124,7,128,56
106 DATA30,0,120,63,0,252,127,129,254,127,129,254,127,189,254,127,255,254
107 DATA63,255,252,31,221,248,3,221,192,1,255,134,3,189,204,1,199,152,1,255,48
108 DATA1,255,224,1,252,0,3,254,0
109 DATA7,14,0,204,14,0,240,56,0,112,112,0,0,60,[illegible]
200 POKES+4,129:POKES+4,128:RETURN
300 POKES+11,129:POKES+11,128:RETURN
```

LINEA 5:

S=54272	Asigna a la variable S el valor 54272, que es la posición de inicio del CHIP DE SONIDO. Desde ahora, siempre que nos refiramos a alguna posición de sonido haremos POKE S más un valor.
POKES+24,15	Igual a POKE 54296,15. Coloca el volumen al máximo.
POKES,220	Igual a POKE 54272,220 que coloca baja frecuencia en la Voz 1 para lograr una nota aproximada a la C alta en la sexta octava.
POKES+1,68	Igual a POKE 54273,68 que coloca la alta frecuencia en la Voz 1 para una nota aproximada a la C alta en la sexta octava.
POKES+5,15	Igual a POKE 54277,15 que coloca el Ataque/Decaimiento para la Voz 1 y en este caso consiste en el máximo nivel de Decaimiento sin Ataque, produciendo efecto de "eco".
POKES+6,215	Igual a POKE 54278,215 que coloca el Sostenimiento/Relajación para la Voz 1 (215 representa una combinación de valores de Sostenimiento/Relajación).

LINEA 10:

POKES+7,120	Igual a POKE 54279,120 que coloca la Alta frecuencia en la Voz 2.
POKES+8,100	Igual a POKE 54280,100 que coloca la Baja frecuencia en la Voz 2.
POKES+12,15	Igual a POKE 54284,15 que coloca el Ataque/Decaimiento en la Voz 2 al mismo nivel que en la Voz 1.
POKES+13,215	Igual a POKE 54285,215 que coloca el Sostenimiento/Relajación en la Voz 2 al mismo nivel que en la Voz 1

LINEA 15:

PRINT "SHIFT CLR/HOME" V=53248	Limpia la pantalla al inicio del programa. Define la variable "V" como la posición de inicio de los registros del VICII, encargado de controlar los Sprites. A partir de ahora, las posiciones referidas a Sprites se mostrarán como V seguido de un número.
POKEV+21,1	Activa el Sprite 1.

LINEA 20:

FORS=12288 TO 12350	Usaremos UN SPRITE (Sprite 0) en esta animación, pero vamos a usar TRES juegos de datos para definir tres formas distintas. Para conseguir la animación, cambiaremos los PUNTEROS del Sprite 0 a tres posiciones distintas de memoria en las que hemos definido tres formas distintas de Sprites. El mismo Sprite puede ser redefinido rápidamente una y otra vez entre las tres distintas formas para producir el efecto de que el ratón está bailando. Usted puede definir docenas de formas distintas para sus Sprites en las instrucciones DATA, y cambiar las formas entre uno o más Sprites. Como puede ver, no tiene la limitación de una forma por Sprite o viceversa. Un Sprite puede tener muchas formas diferentes, cambiando únicamente el PUNTERO DE SPRITE de forma que apunte a las distintas zonas de memoria donde se almacenan las formas de los Sprites. Esta línea significa que hemos colocado los datos para la "forma 1 del Sprite" en las posiciones de memoria de 12288 a 12350
READQ1	Lee 63 números en orden de la lista DATA que empieza en la

línea 100. Q1 es un nombre de variable arbitrario. Podría ser igualmente A, Z1 o cualquier otro nombre de variable numérica.

POKES1,Q1
POKE el primer número de la instrucción DATA en la primera posición de memoria (la primera posición de memoria es 12288). Esto es lo mismo que POKE12288,30.

NEXT
Indica al ordenador que ejecute todas las instrucciones entre FOR y NEXT. En otras palabras, la instrucción NEXT indica al ordenador que lea (READ) el próximo (NEXT) Q1 de la lista DATA, que es 0, y también que incremente S1 con 1 para obtener el siguiente valor, que será 12289. El resultado es POKE12289,0... la instrucción NEXT sigue realizando el bucle hasta que obtiene el último valor de la serie, que en este caso sería POKE 12350,0.

LINEA 25:

FORS2=12352 TO 12414
La segunda forma para el Sprite 0 se define por los datos colocados a partir de la posición 12352 y hasta la posición 12414. Note que se ha [illegible] la posición 12351... ésta es la 64ava posición que se usó en la definición del primer grupo, pero no debe contener ningún valor referente a datos de los Sprites. Recuerde cuando defina Sprites en posiciones sucesivas que debe usar 64 posiciones, pero sólo debe hacer POKE en las 63 primeras.

READ Q2
Lee los 63 números que siguen a los usados para la primera forma. Este READ busca y lee cada vez el siguiente número de la lista DATA.

POKES2,Q2
POKE el valor Q2 en la posición S2 para almacenar la siguiente forma para el Sprite, el almacenamiento empieza en la posición 12352.

NEXT
Mismo uso que en la línea 20.

LINEA 30:

FORS3=12416 TO 12478
La tercera forma para el Sprite 0 se define por los datos contenidos en las posiciones 12416 a 12478.

READQ3
Lee ordenadamente los últimos 63 números de la lista DATA y los asigna uno a uno a la variable Q3.

POKES3,Q3
POKE estos números en las posiciones desde 12416 a 12478.

NEXT
Igual a las líneas 20 y 25.

LINEA 35:

POKEV+39,15
Asigna el color gris claro al Sprite 0.

POKEV+1,68
Coloca la esquina superior izquierda del Sprite en la posición vertical (Y) 68. Para poder comparar, la posición 50 es la más pequeña en la que un Sprite es totalmente visible en la dirección Y (arriba de la pantalla).

LINEA 40:

PRINTTAB([illegible])
Tabula 160 espacios desde el ESPACIO DE CARACTER de la esquina superior izquierda de la pantalla, colocando el cursor

cuatro líneas más abajo desde el comando de limpieza de la pantalla... esto inicia el mensaje PRINT en la sexta línea de la pantalla.

CTRL WHT
Pulse la tecla CTRL y pulse simultáneamente la tecla WHT. Si esto ocurre dentro de unas comillas, aparece un "E invertida". Esto coloca el color de los caracteres a imprimir en BLANCO.

SOY EL RATON BAILARIN
Simple instrucción PRINT.

C= 7
Esto coloca de nuevo el color azul claro cuando termina el mensaje PRINT. Pulse simultáneamente las teclas C= y 7 dentro de comillas, lo que producirá la aparición de un "rombo invertido."

LINEA 45:

P=192
Asigna el valor 192 a la variable P. El número 192 es el puntero que debe usar. En este caso indica que el Sprite 0 tomará la información sobre su forma a partir de la posición 12288. Cambiar este puntero para que indique otra zona de memoria de donde extraer la forma del Sprite es el secreto de usar un solo Sprite para crear animación con tres distintas formas.

LINEA 50:

FORX=0TO347 STEP3
Mueve el Sprite 3 posiciones a la vez (para proporcionar movimiento rápido) desde la posición 0 a la 347.

LINEA 55:

RX=INT(X/256)
RX es el resultado entero de la operación X/256, lo que significa que RX se redondea a cero si X es menor de 256, y RX tiene el valor de 1 cuando X alcanza la posición 256. Usamos RX para efectuar un POKE con su valor en V+16, lo que nos permite acceder a la parte derecha de la pantalla.

LX=X-RX*256
Cuando el Sprite está en la posición cero de X, la fórmula es igual a: LX=0-(0 veces 256) o 0. Cuando el Sprite está en la posición de X 1, la fórmula es: LX=1-(0 veces 256) o 1. Cuando el Sprite está en la posición 256, la fórmula es: LX=256-(1 vez 256) o 0, lo que coloca X nuevamente a 0, con lo que podemos avanzar por la parte derecha de la pantalla gracias a POKE V+16,1.

LINEA 60:

POKEV,LX
Se coloca en V el valor de la posición horizontal (X) del Sprite 0. (Vea la TABLA DE CONFECCION DE SPRITES en la Pág. 176). Como se muestra arriba, el valor LX, que es la posición horizontal del Sprite, cambia de 0 a 255 y después vuelve a cero automáticamente gracias a la ecuación de la línea 55.

POKEV+16,RX
POKE V+16 siempre activa la "parte derecha" de la pantalla cuando se llega a la posición 256 en la pantalla, y se coloca la

posición de nuevo a cero. RX puede ser 0 o 1, basándose en la fórmula para RX desarrollada en la línea 55.

LINEA 70:

IFP=192THEN GOSUB 200	Si el puntero de Sprite contiene el valor 192 (la primera forma para el Sprite) el control de forma de onda para el primer efecto de sonido es colocado a 129 y 128 por la línea 200.

LINEA 75:

IFP=193THEN GOSUB 300	Si el puntero de Sprite contiene el valor 193 (la segunda forma para el Sprite) el control de forma de onda para el segundo efecto de sonido (Voz 2) se coloca en 129 y 128 gracias a la línea 300.

LINEA 80:

POKE2040,P	Coloca en el PUNTERO DE SPRITE el valor 192 (¿recuerda que P=192 en la línea 45? He aquí la aplicación de P).
FORT=1TO60: NEXT	Un simple bucle vacío determina la velocidad de la danza del ratón. (Pruebe con mayor o menor velocidad incrementando o disminuyendo el número 60).

LINEA 85:

P=P+1	Ahora se incrementa el valor del puntero añadiendo 1 al valor original de P.
IFP>194THEN P=A192	Sólo es necesario que el puntero indique 3 posiciones de memoria. 192 apunta a las posiciones 12288 a 12350, 193 apunta a las posiciones 12352 a 12414, y 194 apunta a las posiciones 12416 a 12478. Esta línea pide al ordenador que asigne de nuevo a P su valor original (192) si P tiene el valor de 195, puesto que el programa no puede trabajar con este valor, ya que no se definió la forma de ningún Sprite en las posiciones a las que apuntaría el valor de 195. P es 192, 193, 194 y entonces vuelve otra vez a 192, por lo que el puntero indica consecutivamente las tres formas del Sprite almacenadas en los tres grupos de posiciones de memoria que contienen dichos datos.
NEXTX	Después de que el Sprite tenga una de las tres formas definidas en la lista DATA, es movido a través de la pantalla. Salta 3 posiciones de X a la vez (en lugar de una posición a la vez, lo que también es posible). El salto de tres posiciones a la vez hace que el ratón "baile" más deprisa a través de la pantalla. NEXT X busca el FOR que inició el bucle en la línea 50 para volver a él.

LINEA 95:

END	Finaliza el programa, lo que ocurre cuando el Sprite desaparece por la parte derecha de la pantalla.

LINEAS 100-109

DATA	Las formas del Sprite se toman de los números almacenados en la lista DATA, por orden. Primero los 63 números que componen la forma 1 para el Sprite, después se leen los 63 números siguientes para la forma 2 y por último se leerán los 63 que definen la forma 3. Estos valores se almacenan en tres zonas de memoria, de donde el programa toma la forma del Sprite en cada momento, según a donde apunte el puntero del Sprite 0. Cambiando rápidamente la forma del Sprite se consigue un efecto de animación. Si desea ver cómo los números DATA afectan a la forma del Sprite, pruebe a cambiar los 3 primeros números de la línea 100 por 255, 255, 255. Vea la sección que habla sobre la definición de la forma de los Sprites para más información.

LINEA 200:

POKES+4,129	Control de forma de onda colocado a 129 activa el efecto de sonido.
POKES+4,128	Control de forma de onda colocado a 128 desactiva el efecto de sonido.
RETURN	Hace regresar al programa al final de la línea 70, después de haber cambiado el valor de la forma de onda, para seguir con el programa.

LINEA 300:

POKES+11,129	Control de forma de onda a 129 para activar el efecto de sonido.
POKES+11,128	Control de forma de onda a 128 para desactivar el efecto de sonido.
RETURN	Regresa al final de la línea 75 para continuar con el programa.

TABLA PARA CONFECCIONAR SPRITES FACILMENTE

V = 53.248	SPRITE 0	SPRITE 1	SPRITE 2	SPRITE 3	SPRITE 4	SPRITE 5	SPRITE 6	SPRITE 7
Activar un Sprite	V+21,1	V+21,2	V+21,4	V+21,8	V+21,16	V+21,32	V+21,64	V+21,128
Colocarlo en memoria (Ajustar Punteros)	2040, 192	2041, 193	2042, 194	2043, 195	2044, 196	2045, 197	2046, 198	2047, 199
Posiciones de la forma de los Sprites (12288-12798)	12288 a 12350	12352 a 12414	12416 a 12478	12480 a 12542	12544 a 12606	12608 a 12670	12672 a 12734	12736 a 12798
Color del Sprite	V+39,C	V+40,C	V+41,C	V+42,C	V+43,C	V+44,C	V+45,C	V+46,C
Posicionado X IZQUIERDA Posiciones (0-255)	V+0,X	V+2,X	V+4,X	V+6,X	V+8,X	V+10,X	V+12,X	V+14,X
Posicionado X DERECHA Posiciones (0-255)	V+16,1 V+0,X	V+16,2 V+2,X	V+16,4 V+4,X	V+16,8 V+6,X	V+16,16 V+8,X	V+16,32 V+10,X	V+16,64 V+12,X	V+16,128 V+14,X
Ajuste de la posición Y	V+1,Y	V+3,Y	V+5,Y	V+7,Y	V+9,Y	V+11,Y	V+13,Y	V+15,Y
Expandir Sprites en la dirección horizontal (X)	V+29,1	V+29,2	V+29,4	V+29,8	V+29,16	V+29,32	V+29,64	V+29,128
Expandir Sprites en la dirección vertical (Y)	V+23,1	V+23,2	V+23,4	V+23,8	V+23,16	V+23,32	V+23,64	V+23,128
Activar el Modo Multicolor	V+28,1	V+28,2	V+28,4	V+28,8	V+28,16	V+28,32	V+28,64	V+28,128
Multicolor 1 (Primer color)	V+37,C	V+37,C	V+37,C	V+37,C	V+37,C	V+37,C	V+37,C	V+37,C
Multicolor 2 (Segundo color)	V+38,C	V+38,C	V+38,C	V+38,C	V+38,C	V+38,C	V+38,C	V+38,C
Ajustar prioridades de los Sprites	La regla es que los Sprites con número menor siempre tendrán prioridad sobre los de número superior. Por ejemplo, el Sprite 0 tiene prioridad sobre TODOS los demás Sprites. El Sprite 7 tiene la menor prioridad, pasando siempre por debajo de los demás. Esto significa que los Sprites con número bajo pasarán SIEMPRE por ENCIMA de los de número mayor.							
Colisión (Sprite-Sprite)	V+30 IF PEEK(V+30)ANDX=X THEN (acción)							
Colisión (Sprite-Texto)	V+31 IF PEEK(V+31)ANDX=X THEN (acción)							

NOTAS SOBRE LA CONFECCION DE SPRITES

Punteros de Sprite alternativos y Posiciones de Memoria Usando el Buffer del Cassette

Colocar en memoria (Ajustar punteros)	SPRITE 0 2040,13	SPRITE 1 2041,14	SPRITE 2 2042,15	Si usa de 1 a 3 Sprites puede utilizar estas posiciones de memoria situadas en el buffer del cassette (832 a 1023), pero para más de 3 Sprites, sugerimos utilizar las posiciones a partir de 12288 y hasta 12798 (vea la tabla).
Posiciones de los bloques 13, 14 y 15 conteniendo la forma	832 a 894	896 a 958	960 a 1022	

ACTIVAR SPRITES

Usted puede activar cualquier Sprite individualmente usando POKE V+21 y el número de la tabla... PERO... activar UN SOLO Sprite DESACTIVA todos los demás. Para activar DOS O MAS Sprites, SUME los números de los Sprites que desee activar (Ejemplo: POKE V+21,6 activa los Sprites 1 y 2). He aquí un método para desactivar un Sprite sin afectar a los demás (útil para animación).

EJEMPLO:

Para desactivar el Sprite 0 escriba: POKE V+21,PEEK(V+21)AND(255-1). Cambie el número 1 en (255–1) por 1, 2, 4, 8, 16, 32, 64, o 128 (para los Sprites 0 a 7). Para reactivar un Sprite sin afectar a los demás, teclee: POKE V+21,PEEK (V+21)OR 1, y cambie el OR 1 por OR 2 (Sprite 2) OR 4 (Sprite 3) etc.

POSICIONES DE X SUPERIORES A 255:

Las posiciones de X van de 0 a 255... y entonces EMPIEZAN OTRA VEZ de 0 a 255. Para colocar un Sprite en una posición mayor de 255 –a la derecha de la pantalla– debe primero hacer POKE V+16 como se muestra en la tabla, entonces POKE un nuevo valor de X entre 0 y 63, lo que coloca al Sprite en la parte derecha de la pantalla. Para volver a las posiciones 0–255, POKE V+16,0 y POKE un valor de X entre 0 y 255.

VALORES DE LA POSICION Y:

Las posiciones de Y van de 0 a 255, incluyendo 0 a 49, fuera de la pantalla por la parte de arriba, 50–229 en la parte visible de la pantalla, y 230–255 fuera de la pantalla por la parte de abajo.

COLORES DE LOS SPRITES

Para presentar el Sprite 0 BLANCO, escriba: POKE V+39,1 (Use el POKE de color determinado para cada Sprite mostrado en la tabla y el código de color mostrado aquí):

0-NEGRO	4-PURPURA	8 NARANJA	12-GRIS MEDIO
1-BLANCO	5-VERDE	9-MARRON	13-VERDE CLARO
2-ROJO	6-AZUL	10-ROJO CLARO	14-AZUL CLARO
3-CYAN	7-AMARILLO	11-GRIS OSCURO	15-GRIS CLARO

POSICION DE MEMORIA:

Usted debe reservar un bloque de 64 bytes de memoria por cada Sprite que requiera, de los que 63 serán usados para almacenar la forma del Sprite. Las posiciones de memoria mostradas más abajo son las recomendadas para los "punteros de Sprite" de la tabla. Cada Sprite es único y debe ser definido. Sin embargo, si desea varios Sprites iguales, coloque los punteros de los Sprites que desee con el mismo valor. Así todos buscarán la información sobre forma en el mismo sitio y por lo tanto serán iguales.

DIFERENTES POSICIONES DE LOS PUNTEROS DE SPRITES:

Estas posiciones son UNICAMENTE RECOMENDADAS.
Precaución: usted puede colocar los punteros de Sprite de forma que señalen cualquier parte de la RAM existente, pero si coloca los punteros –y por lo tanto las definiciones de sus Sprites– en una zona de memoria "baja", un programa largo escrito en BASIC puede invadir la zona de datos, o viceversa. Para proteger un programa ESPECIALMENTE LARGO en BASIC del área para datos, debe colocar las definiciones de los Sprites en una parte "alta" de la memoria (por ejemplo, 2040,192 para el Sprite 0, en las posiciones 12288 a 12350... 2041,193 en las posiciones 12352-12414 para el Sprite 1, etc.). Ajustando las posiciones de memoria de donde los Sprites toman la forma, usted puede definir hasta 64 Sprites diferentes con un programa normal en BASIC. Para lograr esto, defina las formas que desee en una lista de instrucciones DATA y luego redefina un Sprite particular cambiando el valor de su puntero, para que indique otra zona de memoria. Vea el "Ratón Bailarín" para comprender cómo funciona esta técnica. Si desea que dos o más Sprites tengan la MISMA FORMA (puede cambiar el color y la posición de cada Sprite), use el mismo valor del puntero para todos los Sprites que desee que sean iguales. (Por ejemplo, puede usar las mismas posiciones de memoria para los Sprites 0 y 1 mediante POKE 2040,192 y POKE 2041,192).

PRIORIDAD:

La prioridad significa que un Sprite puede moverse "por delante" o "por detrás" de otro Sprite en la pantalla. Los Sprites con mayor prioridad se mueven "por encima" de los Sprites de prioridad inferior. La regla es que los Sprites con número inferior tienen prioridad sobre los de un número más alto. El Sprite 0 tiene prioridad sobre todos los demás. El Sprite 7 no tiene prioridad sobre ninguno. El Sprite 1 tiene prioridad sobre los Sprites 2–7, etc. Si usted coloca dos Sprites en la misma posición, el Sprite con mayor prioridad aparece ENCIMA o POR DELANTE del Sprite con prioridad inferior. El Sprite con prioridad excepto en las zonas "transparentes" de este último, por las que se ve parte del Sprite de debajo.

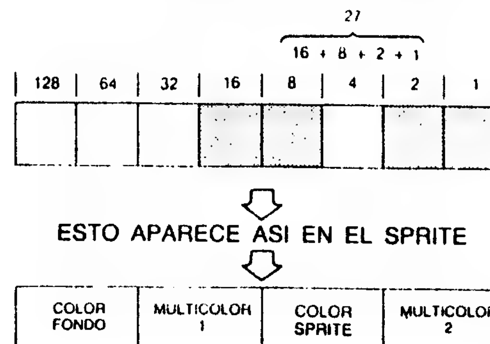
USO DEL MODO MULTICOLOR

Usted puede crear Sprites multicolores aunque el uso del modo multicolor requiere PAREJAS de puntos en lugar de puntos individuales en el dibujo de su Sprite. (En

otras palabras, cada punto coloreado en su Sprite consiste en dos puntos juntos). Usted tiene cuatro colores para escoger de: Color del Sprite (tabla anterior), Multicolor 1, Multicolor 2 y "color del fondo" (el color del fondo es conseguido con la pareja de puntos desactivados (a 0), y los puntos aparecen transparentes). Considere un bloque horizontal de 8 puntos en el dibujo de un Sprite. El color de CADA PAR de puntos se determina mediante las combinaciones posibles iluminado/apagado de cada pareja, según la siguiente tabla:

COLOR DEL FONDO	(Con los dos puntos en blanco, se muestra el color existente debajo del Sprite).
MULTICOLOR 1	(Iluminando el punto de la derecha en un par de puntos, LOS DOS PUNTOS se mostrarán del color Multicolor 1).
COLOR DE SPRITE	(Iluminando el punto de la izquierda en un par de puntos, LOS DOS PUNTOS aparecerán del color del Sprite).
MULTICOLOR 2	(Iluminando los dos puntos, AMBOS se mostrarán del color Multicolor 2).

Mire la línea horizontal de 8 puntos mostrada más abajo. Este bloque coloca el color del fondo en los dos primeros puntos, los segundos dos puntos serán del color Multicolor 1, la tercera pareja de puntos será del color del Sprite, y por último, la cuarta pareja de puntos será del color Multicolor 2. El color de cada PAREJA de puntos depende de que bits de cada pareja están activados y cuáles desactivados, de acuerdo con la ilustración anterior. Después de determinar el color deseado para cada pareja de puntos, el próximo paso es sumar los valores de los puntos iluminados en cada bloque de 8 puntos, y POKE el resultado en la posición de memoria adecuada. Por ejemplo, si la línea de 8 puntos mostrada abajo es el primer bloque en un Sprite que se inicia en la posición 832, el valor de los puntos iluminados es 16+8+2+1 = 27, por lo que efectuaremos POKE 832,27



COLISION:

Usted puede detectar cuando un Sprite ha chocado con otro usando esta línea: IF PEEK(V+30)ANDX=XTHEN (inserte aquí una acción). Esta línea detecta si un

Sprite particular ha chocado CON CUALQUIER OTRO SPRITE, donde X es igual a 1 para el Sprite 0, 2 para el Sprite 1, 4 para el Sprite 2, 8 para el Sprite 3, 16 para el Sprite 4, 32 para el Sprite 5, 64 para el Sprite 6, y 128 para el Sprite 7. Para comprobar si un Sprite ha chocado con un "CARACTER DE LA PANTALLA" use esta línea: IF PEEK(V+31)ANDX=XTHEN (Inserte aquí la acción).

USO DE CARACTERES GRAFICOS EN LAS INSTRUCCIONES DATA

El siguiente programa crea un Sprite usando espacios y círculos sólidos (SHIFT Q) en las instrucciones DATA. El Sprite y los números POKE que lo forman son mostrados en la pantalla.

SHIFT CLR/HOME

```
10 PRINT"♥":FORI=0TO63:POKE832+I,0:NEXT
20 GOSUB60000
999 END
60000 DATA"         ●●●●●●●        "
60001 DATA"       ●●●●●●●●●●●      "
60002 DATA"      ●●●●●●●●●●●●●     "
60003 DATA"      ●●●●●   ●●●●●     "
60004 DATA"     ●●●●● ●●●  ●●●●    "
60005 DATA"     ●●●●● ●●● ●●●●●    "
60006 DATA"     ●●●●● ●●●  ●●●●    "
60007 DATA"      ●●●●●   ●●●●●     "
60008 DATA"      ●●●●●●●●●●●●●     "
60009 DATA"      ●●●●●●●●●●●●●     "
60010 DATA"      ● ●●●●●●●●● ●     "
60011 DATA"       ● ●●●●●●● ●      "
60012 DATA"       ●  ●●●●●  ●      "
60013 DATA"        ●  ●●●  ●       "
60014 DATA"        ●  ●●●  ●       "
60015 DATA"         ●  ●  ●        "
60016 DATA"         ●  ●  ●        "
60017 DATA"          ●●●●●         "
60018 DATA"          ●●●●●         "
60019 DATA"          ●●●●●         "
60020 DATA"           ●●●          "
60100 V=53248:POKEV,200:POKEV+1,100:POKEV+21,1:POKEV+39,14:POKE2040,13
60105 POKEV+23,1:POKEV+29,1
60110 FORI=0TO20:READA$:FORK=0TO2:T=0:FORJ=0TO7:B=0
60140 IFMID$(A$,J+K*8+1,1)="●"THENB=1
60150 T=T+B*2↑(7-J):NEXT:PRINTT;:POKE832+I*3+K,T:NEXT:PRINT:NEXT
60200 RETURN
```


CAPITULO 4

PROGRAMACION DE SONIDO Y MUSICA EN SU COMMODORE 64

INTRODUCCION

Su Commodore 64 está equipado con uno de los más sofisticados generadores de sonido de los disponibles en cualquier ordenador. Posee tres voces totalmente programables, **ATAQUE/DECAIMIENTO/SOSTENIMIENTO/RELAJACION (ADSR)**, filtros, modulación, y "ruido blanco". Todas estas características están disponibles directa y fácilmente mediante instrucciones BASIC y/o lenguaje maquina. Esto significa que usted puede crear sonidos y musica muy compleja usando programas relativamente fáciles de diseñar.

Esta sección del Manual de Referencia del Programador ha sido creada para ayudarle a explorar todas las características del chip de sonido 6581 "SID", el sintetizador de sonido y música incorporado a su ordenador. Explicaremos la teoría, ideas musicales y la forma de convertir estas ideas en canciones completas introducidas en su ordenador.

Usted no necesita ser un experto programador o un excelente músico para aprovechar las habilidades musicales de su Commodore 64. Esta sección está llena de ejemplos de programación con completas explicaciones para que pueda iniciarse en las técnicas adecuadas.

El generador de sonido es activado mediante POKE en posiciones de memoria específicas. Una completa lista de estas posiciones se encuentra en el Apéndice O. Deseamos explicarle cada concepto, paso a paso. Al finalizar el estudio, usted será capaz de crear una variedad casi infinita de sonidos, y estará listo para realizar sus propios experimentos de sonido y música.

Cada sección de este capítulo se inicia con un pequeño programa de ejemplo y una completa explicación línea a línea de lo que el mismo realiza, para mostrarle el modo de funcionamiento y las características de cada función. Las explicaciones técnicas son para que las lea siempre que tenga curiosidad por saber las teorías de la formación de sonido.

La herramienta de trabajo de los programas de sonido es la instrucción POKE. POKE coloca un valor determinado (NUM) en una posición de memoria específica. (MEM)

POKE MEM,NUM

Las posiciones de memoria (MEM) empleadas para la sintetización musical empiezan en la 54272 y terminan en la 54296, ambas incluidas. Estas son las posiciones con las que debe trabajar la instrucción POKE para usar el chip 6581 (SID). Otra forma de usar las posiciones mencionadas antes es recordar solo la posición 54272 y entonces sumarle un numero entre 0 y 27. Haciendo esto usted puede hacer POKE en todas las posiciones (de 54272 a 54296) que necesita del chip SID. Los números (NUM) que debe usar en sus instrucciones POKE deben estar entre 0 y 255, ambos incluidos.

Cuando usted tenga un poco de práctica en la confección de música, puede introducirse más a fondo en las técnicas de programación usando la función PEEK. PEEK es una función que lee el contenido de una posición de memoria específica.

X=PEEK(MEM)

El valor de la variable X es el contenido actual de la posición de memoria MEM. Por supuesto, sus programas incluirán otras instrucciones BASIC, pero para una completa explicación de las mismas debe consultar la sección de Instrucciones BASIC de este manual. (Pág. 23 y siguientes).

Ahora pruebe este simple programa que utiliza únicamente 1 de las 3 voces disponibles. El ordenador ¿está listo? Escriba NEW, entre este programa y ejecútelo (RUN). Después guárdelo en su Commodore DATASSETTE (R) o en su unidad de disco Commodore.

PROGRAMA DE EJEMPLO 1:

```
5 S=54272
10 FORL=STOS+24:POKEL,0:NEXT:REM BORRA EL CHIP DE SONIDO
20 POKES+5,9:POKES+6,0
30 POKES+24,15:REM PONE EL VOLUMEN AL MAXIMO
40 READHF,LF,DR
50 IFHF<0THENEND
60 POKES+1,HF:POKES,LF
70 POKES+4,33
80 FORT=1TODR:NEXT
90 POKES+4,32:FORT=1TO50:NEXT
100 GOTO40
110 DATA25,177,250,28,214,250
120 DATA25,177,250,25,177,250
130 DATA25,177,125,28,214,125
140 DATA32,94,750,25,177,250
150 DATA28,214,250,19,63,250
160 DATA19,63,250,19,63,250
170 DATA21,154,63,24,63,63
180 DATA25,177,250,24,63,125
190 DATA19,63,250,-1,-1,-1
```

A continuación se explica el programa línea a línea. Estúdiela junto con el programa, hasta comprender que es lo que hace cada instrucción del programa.

EXPLICACION LINEA A LINEA DEL PROGRAMA 1

Lineas	Descripción
5	Asigna a S el valor del inicio del chip de sonido
10	Borra todos los registros de sonido
20	Asigna Ataque/Decaimiento para la voz 1 (A=0, D=9).
	Asigna Sostenimiento/Relajación a la voz 1 (S=0, R=0).
30	Coloca el volumen al máximo.
40	Lee la alta y baja frecuencia y la duración de la nota.
50	Si la frecuencia es menor de 0 la canción se ha terminado.
60	POKE las frecuencias alta y baja en la voz 1.
70	Coloca la forma de onda "diente de sierra" para la voz 1.
80	Bucle de tiempo para la duración de la nota.
90	Desactiva la forma de onda "diente de sierra" para la voz 1.
100 180	Regreso para ejecutar la siguiente nota.
	Datos de la canción: alta frecuencia, baja frecuencia, duración de la nota.
190	Ultima nota de la canción y −1 señalando el final de la canción.

CONTROL DE VOLUMEN

El registro 24 del chip contiene el control de volumen. El volumen puede colocarse entre 0 y 15. Los otros 4 bits se usan para propósitos que veremos más adelante. Por ahora es suficiente que conozca que el volumen va de 0 a 15. Mire la línea 30 del programa 1 para ver cómo es seleccionado.

FRECUENCIAS DE LAS ONDAS SONORAS

El sonido es creado por el movimiento de ondas en el aire. Piense en una piedra lanzada al agua y las ondas que forma. Unas ondas similares son creadas en el aire para generar sonidos. Si cuenta el tiempo trascurrido entre el pico de una onda y el de la siguiente, encontrará el número de segundos por ciclo en la onda (n=número de segundos). El recíproco de este número (1/n) le da el número de ciclos por segundo. Los ciclos por segundo se conocen comúnmente como la frecuencia. La altura de un sonido (grado de elevación) se determina por la frecuencia de las ondas que lo producen.

El generador de sonido de su Commodore usa dos posiciones para determinar la frecuencia. El Apéndice E le proporciona los valores de las frecuencias para reproducir ocho octavas completas de notas musicales. Para crear una frecuencia de las que no están especificadas en la lista use "F_{out}"(frecuencia de salida) y la siguiente fórmula para representar la frecuencia (F_n) del sonido que desee crear. Recuerde que cada nota requiere un número de alta y baja frecuencia.

$$F_n = F_{out}/.06097$$

Una vez haya averiguado el valor de F_n para su "nueva" nota el próximo paso es

crear los valores de alta y baja frecuencia para su nota. Para lograr esto, debe redondear el valor de F despreciando el valor decimal. Usted tiene ahora un valor entero. Ahora puede descubrir la alta frecuencia de la nota (F_{hi}) usando la fórmula $F_{hi} = F_n/256$ y la baja frecuencia (F_{lo}) mediante la fórmula $F_{lo} = F_n-(256*F_{hi})$.

En este punto usted ya ha ejecutado una canción con una voz de su ordenador. Si desea detenerse aquí sustituya los números DATA por los de su canción favorita y será el "director" de la "orquesta electrónica" en su sala de conciertos casera.

USO DE VOCES MULTIPLES

Su ordenador Commodore tiene tres voces (osciladores) controladas independientemente. Nuestro primer ejemplo sólo utilizaba una de ellas. Más adelante, usted aprenderá a cambiar la calidad del sonido creado por las voces. Pero ahora vamos a ver cómo tocan las tres voces juntas.

Este programa de ejemplo le muestra una forma de trasladar una partitura a su "orquesta electrónica". Escríbalo, y guárdelo (SAVE) en su DATASSETTE (R) o en su unidad de disco Commodore. No olvide ejecutar NEW antes de introducir el programa.

PROGRAMA DE EJEMPLO 2:

```
10 S=54272:FORL=STOS+24:POKEL,0:NEXT
20 DIMH(2,200),L(2,200),C(2,200)
30 DIMFQ(11)
40 V(0)=17:V(1)=65:V(2)=33
50 POKES+10,8:POKES+22,128:POKES+23,244
60 FORI=0TO11:READFQ(I):NEXT
100 FORK=0TO2
110 I=0
120 READNM
130 IFNM=0THEN250
140 WA=V(K):IFNM<0THENNM=-NM:WA=1
150 DR%=NM/128:OC%=(NM-128*DR%)/16
160 NT=NM-128*DR%-16*OC%
170 FR=FQ(NT)
180 IFOC%=7THEN200
190 FORJ=6TOOC%STEP-1:FR=FR/2:NEXT
200 HF%=FR/256:LF%=FR-256*HF%
210 IFDR%=1THENH(K,I)=HF%:L(K,I)=LF%:C(K,I)=WA:I=I+1:GOTO 120
220 FORJ=1TODR%-1:H(K,I)=HF%:L(K,I)=LF%:C(K,I)=WA:I=I+1:NEXT
230 H(K,I)=HF%:L(K,I)=LF%:C(K,I)=WA-1
240 I=I+1:GOTO120
250 IFI>IMTHENIM=I
260 NEXT
500 POKES+5,0:POKES+6,240
```

```
510 POKES+12,85:POKES+13,133
520 POKES+19,10:POKES+20,197
530 POKES+24,31
540 FORI=0TOIM
550 POKES,L(0,I):POKES+7,L(1,I):POKES+14,L(2,I)
560 POKES+1,H(0,I):POKES+8,H(1,I):POKES+15,H(2,I)
570 POKES+4,C(0,I):POKES+11,C(1,I):POKES+18,C(2,I)
580 FORT=1TO80:NEXT:NEXT
590 FORT=1TO200:NEXT:POKES+24,0
600 DATA34334,36376,38539,40830
610 DATA43258,45830,48556,51443
620 DATA54502,57743,61176,64814
1000 DATA594,594,594,596,596
1010 DATA1618,587,592,587,585,331,336
1020 DATA1097,583,585,585,585,587,587
1030 DATA1609,585,331,337,594,594,593
1040 DATA1618,594,596,594,592,587
1050 DATA1616,587,565,331,336,841,327
1060 DATA1607
1999 DATA0
2000 DATA583,585,583,583,327,329
2010 DATA1611,583,585,578,578,578
2020 DATA196,198,583,326,578
2030 DATA326,327,329,327,329,326,578,583
2040 DATA1606,582,322,324,582,587
2050 DATA329,327,1606,583
2060 DATA327,329,587,331,329
2070 DATA329,328,1609,578,834
2080 DATA324,322,327,585,1602
2999 DATA0
3000 DATA567,566,567,304,306,308,310
3010 DATA1591,567,311,310,567
3020 DATA306,304,299,308
3030 DATA304,171,176,306,291,551,306,308
3040 DATA310,308,310,306,295,297,299,304
3050 DATA1586,562,567,310,315,311
3060 DATA308,313,297
3070 DATA1586,567,560,311,309
3080 DATA308,309,306,308
3090 DATA1577,299,295,306,310,311,304
3100 DATA562,546,1575
3999 DATA0
```


He aquí la explicación línea a línea del programa de ejemplo 2. Por ahora, interésese en cómo se controlan las tres voces.

EXPLICACION LINEA A LINEA DEL PROGRAMA 2:

Línea (s)	Descripción
10	Asigna el inicio de las posiciones de sonido a S y borra los registros de sonido.
20	DIMensiona tablas para contener la actividad de la canción, 1/16 de medida por posición.
30	DIMensiona una tabla para contener la frecuencia de cada nota.
40	Almacena el control de onda para cada voz.
50	Asigna la anchura de pulso para la voz 2. Asigna la alta frecuencia para el filtrado. Asigna la resonancia para el filtro y el filtro de la tercera voz.
60	Lee la frecuencia base de cada nota.
100	Inicia el bucle de decodificación para cada voz.
110	Inicializa el puntero de la tabla de actividad.
120	Lee el código de nota.
130	Si el código es 0, pasa a la siguiente voz.
140	Coloca la forma de onda adecuada a la voz. Si es un silencio, el control de onda está a 1.
150	Decodifica duración y octava.
160	Decodifica nota.
170	Asigna la frecuencia de base para esta nota.
180	Si es la octava alta, salta el bucle de división.
190	Divide la frecuencia de base por 2 el número de veces necesario.
200	Asigna la alta y baja frecuencia.
210	Si es la dieciseisava nota, asigna tabla: alta frecuencia, baja frecuencia, y control de onda.
220	Para el último impulso de la nota asigna tabla: alta frecuencia, baja frecuencia, control de onda. (Voz activada).
230	Para el último pulso de la nota asigna tabla: alta frecuencia, baja frecuencia, control de onda. (Voz desactivada).
240	Incrementa el puntero de la tabla. Busca la siguiente nota.
250	Si es más larga que antes, reasigna número de actividades.
260	Pasa a la siguiente voz.
500	Asigna Ataque/Decaimiento para voz 1 (A=0, D=0). Asigna Sostenimiento/Relajación voz 1 (S=15, R=0).
510	Asigna Ataque/Decaimiento voz 2 (A=5, D=5) Asigna Sostenimiento/Relajación voz 2 (S=8, R=5)
520	Asigna Ataque/Decaimiento voz 3 (A=0, D=10) Asigna Sostenimiento/Relajación voz 3 (S=12, R=5)
530	Volumen a 15, filtro pasabajo.
540	Inicia bucle para cada 1/16 de medida.

Línea (s)	Descripción
550	POKE baja frecuencia de la tabla para todas las voces.
560	POKE alta frecuencia de la tabla para todas las voces.
570	POKE control de forma onda de la tabla para todas las voces.
580	Bucle de tiempo para cada 1/16 de media y retorno para la próxima 1/16 de medida.
590	Pausa. Después apaga volumen.
600–620	Datos de la frecuencia base.
1000–1999	Datos voz 1.
2000–2999	Datos voz 2.
3000–3999	Datos voz 3.

Los valores usados en las instrucciones DATA se han extraído de la tabla del Apéndice E y de la siguiente tabla:

TIPO DE NOTA	DURACION
1/16	128
1/8	256
PUNTEADA 1/8	384
1/4	512
1/4+1/16	640
PUNTEADA 1/4	768
1/2	1024
1/2+1/16	1152
1/2+1/8	1280
PUNTEADA 1/2	1536
ENTERA	2048

El número de nota de la tabla de notas se suma a la duración según la tabla anterior. Entonces, cada nota puede ser entrada usando sólo un número que es decodificado por el programa. Este es sólo uno de los métodos para codificar los valores de las notas. Usted puede usar el que le sea más cómodo. El sistema usado en este programa para codificar notas es el siguiente:

1) La duración (número de medidas 1/16) es multiplicada por 8.
2) El resultado del paso 1 se suma a la octava elegida (0–7).
3) El resultado del paso 2 es multiplicado por 16.
4) Sume el valor de la nota elegida (0–11) al resultado del paso 3.

En otras palabras:

$$((((D*8)+O)*16)+N)$$

Donde D=duración, O=octava, y N=nota.
Se obtiene un silencio usando el negativo del número de duración (Número de medidas 1/16 * 128).

CONTROL DE MULTIPLES VOCES

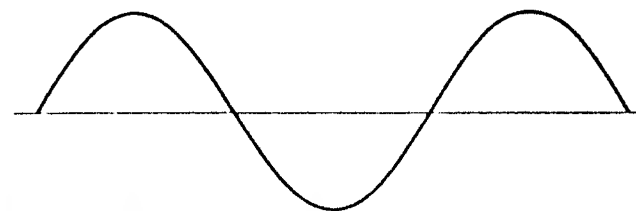
Una vez haya visto el uso de las tres voces, sabrá que el ritmo de las distintas voces debe ser coordinado. Esto se realiza en el programa de la siguiente forma:

1) Dividir cada medida musical en 16 partes.
2) Almacenar lo que ocurre en cada intervalo de 1/16 en tres tablas distintas.

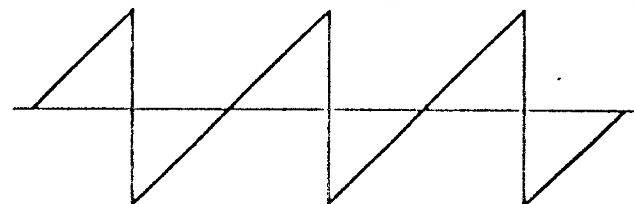
Las frecuencias alta y baja se calculan dividiendo las frecuencias de la octava más alta por dos (líneas 180–190). El control de onda es una señal para iniciar la nota o continuarla cuando ya está sonando. Es también una señal de stop para detener la ejecución de la nota. La elección de forma de onda para cada voz se realiza en la línea 40.
También, éste es sólo uno de los métodos para controlar voces múltiples. Usted puede probar sus propios métodos. Sin embargo, usted puede ahora tomar cualquier partitura y asignar el valor apropiado a cada nota para poder ejecutar melodías con tres voces.

CAMBIO DE LA FORMA DE ONDA

La calidad tonal del sonido se llama timbre. El timbre de un sonido es determinado en principio por su "forma de onda". Si usted recuerda el ejemplo de la piedra en el agua sabrá que las ondas forman circunferencias alrededor del punto de impacto. Estas ondas son casi iguales a las primeras que vamos a estudiar, las ondas sinusoides o senoidales (mostradas abajo).



Para obtener un poco más de práctica, vuelva al primer programa de ejemplo para investigar las distintas formas de onda. La razón de que le hagamos retroceder al primer programa es que es más fácil efectuar los cambios con sólo una voz. Cargue (LOAD) el primer programa de música de su DATASSETTE (R) o su unidad de disco Commodore y ejecútelo (RUN) otra vez. Este programa utiliza la forma de onda "diente de sierra" (mostrada aquí).

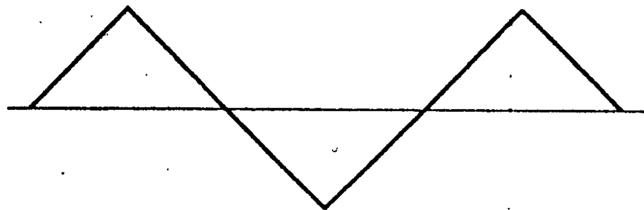


Pruebe a cambiar el número de la línea 70 de 33 a 17 y el número de la línea 90 de 32 a 16. Su programa aparecerá de la siguiente forma:

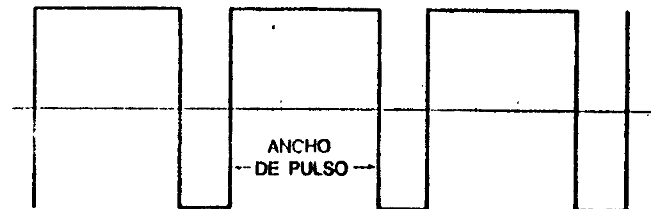
PROGRAMA DE EJEMPLO 3 (EJEMPLO 1 MODIFICADO)

```
5 S=54272
10 FORL=STOS+24:POKEL,0:NEXT
20 POKES+5,9:POKES+6,0
30 POKES+24,15
40 READHF,LF,DR
50 IFHF<0THENEND
60 POKES+1,HF:POKES,LF
70 POKES+4,17
80 FORT=1TODR:NEXT
90 POKES+4,16:FORT=1TO50:NEXT
100 GOTO40
110 DATA25,177,250,28,214,250
120 DATA25,177,250,25,177,250
130 DATA25,177,125,28,214,125
140 DATA32,94,750.25,177,250
150 DATA28,214,250,19,63,250
160 DATA19,63,250,19,63,250
170 DATA21,154,63,24,63,63
180 DATA25,177,250,24,63,125
190 DATA19,63,250,-1,-1,-1
```

Ahora ejecute (RUN) el programa.
Advierta que la calidad del sonido es distinta, menos vibrante, más hueco. Esto ocurre porque hemos cambiado la forma de onda a una onda triangular (mostrada abajo).



La tercera forma de onda musical se llama onda de pulso variable (mostrada abajo).



Esta es una onda rectangular y usted determina la longitud del ciclo de pulso de la onda que desee. Esto se realiza para la voz número 1 usando los registros dos y tres. El registro 2 es el byte menos significativo de la anchura de pulso (L_{pw} = 0 a 255). El registro 3 contiene los cuatro bits más significativos. (H_{ow} = 0 a 15).

Estos dos registros juntos especifican un número de 12 bits para el ancho de pulso de la onda, que puede determinar usando la siguiente fórmula:

$$PW_n = H_{pw}*256 + L_{pw}$$

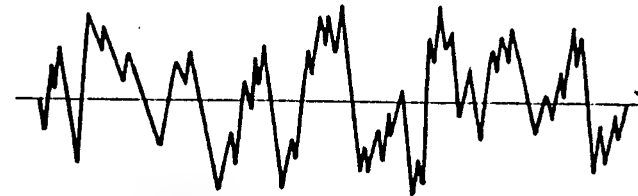
La anchura de pulso se determina mediante la siguiente ecuación:

$$PW_{out} = (PW_n/40.95)\%$$

Donde si PW tiene un valor de 2048, proporciona una onda cuadrada. Esto significa que el registro 2 (L_{pw})=0 y el registro 3 (H_{pw})=8.
Ahora incluya esta línea en su programa:

```
15 POKES+3,8:POKES+2,0
```

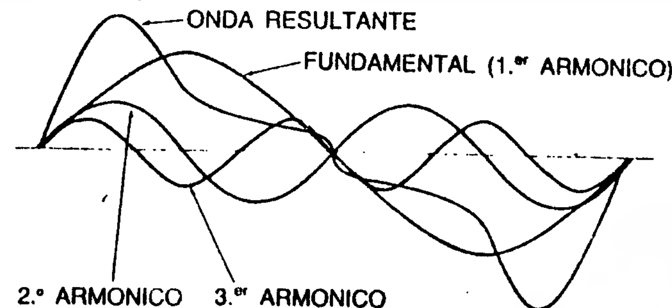
Ahora cambie el número de inicio en la línea 70 de 17 a 65 y el número de paro de la línea 90 de 16 a 64, y ejecute (RUN) el programa. Cambie ahora el ancho de pulso (registro 3 en línea 15) de 8 a 1. ¿Advierte el dramático cambio en la calidad del sonido?
La última forma de onda disponible para usted es el ruido blanco (Mostrada abajo).



Esta forma de onda se usa con frecuencia para lograr efectos de sonido especiales. Para oír cómo suena, cambie el número de inicio de la línea 70 de 65 a 129 y el número de paro de la línea 90 de 64 a 128.

COMPRENSION DE LAS FORMAS DE ONDA

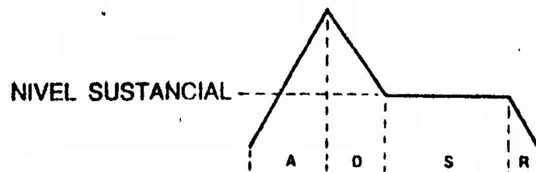
Cuando una nota suena, consiste en una onda que oscila en una frecuencia fundamental y los armónicos de dicha onda.
La frecuencia fundamental define el grado de elevación de la nota. Los armónicos son ondas con frecuencias que son múltiplos enteros de la frecuencia fundamental. Una onda sonora se compone de la frecuencia fundamental y todos los armónicos que forman el sonido.



La teoría musical dice que la frecuencia fundamental es el armónico número 1. El segundo armónico tiene una frecuencia doble de la fundamental. El tercer armónico es tres veces la frecuencia fundamental, y así sucesivamente. La suma de armónicos presentes en la nota proporcionan el timbre de la misma.
Un instrumento acústico, como la guitarra o el violín, tiene una estructura de armónicos muy complicada. De hecho, la estructura de los armónicos puede variar en una misma nota. Usted ya ha probado las formas de onda disponibles en el sintetizador de música de su Commodore. Ahora vamos a hablar acerca de cómo los armónicos trabajan con las ondas triangulares, de diente de sierra y rectangulares.
Una onda triangular contiene solo armónicos impares. El valor de cada armónico presente es proporcional al recíproco del cuadrado del número de armónico. En otras palabras, el número de armónico 3 es 1/9 más bajo que el armónico 1, porque el cuadrado de 3 es 9 (3X3) y el recíproco de 9 es 1/9.
Como puede ver, hay una similitud en la forma de la onda triangular con una onda senoidal oscilando en la frecuencia fundamental.
Las ondas de diente de sierra contienen todos los armónicos. El valor de cada armónico presente es proporcional al recíproco del número de armónico. Por ejemplo, el armónico 2 es 1/2 ruidoso que el armónico número 1.
La onda rectangular contiene armónicos impares en proporción al recíproco del número de armónico. Otras ondas rectangulares contienen armónicos variables. Cambiando el ancho de pulso, el timbre del sonido producido por una onda rectangular puede variar tremendamente.
Escogiendo con cuidado la forma de onda a usar, usted puede conseguir una estructura de armónicos que se parezca mucho al sonido que desee. Para refinar el sonido puede añadir otro aspecto de la calidad de sonido disponible en su Commodore 64, que se llama filtro. Esto se explicará más adelante.

EL GENERADOR DE ENVOLVENTE

El volumen de un tono musical cambia desde el momento en que empieza a oírlo hasta que desaparece el sonido. Cuando una nota es producida, va desde el volumen 0 al volumen máximo. El tiempo que tarda en conseguirlo se llama **ATAQUE**. Después, la nota pasa del volumen máximo a un volumen medio. El tiempo en que ocurre esto se llama **DECAIMIENTO**. El propio volumen medio se llama el volumen de **SOSTENIMIENTO**, hasta que por fin vuelve al volumen cero. El tiempo en que tarda en conseguirlo se llama **RELAJACION**. A continuación se muestra un diagrama de las fases de una nota:



Cada una de las fases mencionadas antes, da una serie de calidades y restricciones a la nota. Los límites se llaman parámetros.
Los parámetros de **ATAQUE/DECAIMIENTO/ SOSTENIMIENTO/RELAJACION** llamados también **ADSR**, pueden ser controlados para su uso por otro conjunto de posiciones del chip generador de sonido. LOAD el primer programa de ejemplo otra vez. Ejecútelo (RUN) y recuerde cómo suena. Entonces, pruebe a cambiar la línea 20 de forma que el programa aparezca como sigue:

PROGRAMA DE EJEMPLO 4 (EJEMPLO 1 MODIFICADO):

```
5 S=54272
10 FORL=STOS+24:POKEL,0:NEXT
20 POKES+5,88:POKES+6,195
30 POKES+24,15
40 READHF,LF,DR
50 IFHF<0THENEND
60 POKES+1,HF:POKES,LF
70 POKES+4,33
80 FORT=1TODR:NEXT
90 POKES+4,32:FORT=1TO50:NEXT
100 GOTO40
110 DATA25,177,250,28,214,250
120 DATA25,177,250,25,177,250
130 DATA25,177,125,28,214,125
140 DATA32,94,750,25,177,250
150 DATA28,214,250,19,63,250
160 DATA19,63,250,19,63,250
170 DATA21,154,63,24,63,63
180 DATA25,177,250,24,63,125
190 DATA19,63,250,-1,-1,-1
```

Los registros 5 y 6 definen el ADSR para la voz 1. El **ATAQUE** es la mitad alta de registro 5, es decir los cuatro bits más significativos del registro 5. El **DECAIMIENTO** está contenido en la mitad baja del registro 5. Usted puede escoger cualquier número entre 0 y 15 para el **ATAQUE**, multiplicarlo por 16 y añadir cualquier numero entre 0 y 15 para el **DECAIMIENTO**. Los valores que corresponden a estos números se muestran mas adelante.
El nivel de **SOSTENIMIENTO** se encuentra en los cuatro bits más significativos del registro 6. Puede ir de 0 a 15. Define el volumen de sostenimiento para la nota. El tiempo de **RELAJACION** se encuentra en los otros 4 bits del registro 6.

He aquí el significado de los valores de ATAQUE, DECAIMIENTO, SOSTENIMIENTO Y RELAJACION:

VALOR	TIEMPO DE ATAQUE (TIEMPO/CICLO)	TIEMPO DE DECAIMIENTO/RELAJACION (TIEMPO/CICLO)
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 s
11	800 ms	2.4 s
12	1 s	3 s
13	3 s	9 s
14	5 s	15 s
15	8 s	24 s

He aquí unos valores para colocar en el programa de ejemplo. Pruebe con otros valores. La variedad de sonidos que puede producir es inmensa. Para lograr un sonido parecido al del violín, cambie la línea 20 por la que sigue:

```
20 POKES+5,88.POKES+6,89:REM A=5;D=8;S=5;R=9
```

Cambie a la forma de onda triangular y descubra el sonido de un xilófono usando estas líneas:

```
20 POKES+5,9:POKES+6,9:REM A=0;D=9;S=0;R=9
70 POKES+4,17
90 POKES+4,16:FOR T=1 TO 50:NEXT
```

Cambie a la forma de onda cuadrada y pruebe el sonido de un piano con las siguientes líneas:

```
15POKES+3,8:POKES+2,0
20POKES+5,9:POKES+6,0:REM A=0;D=9;S=0;R=0
70POKES+4,65
90POKES+4,64:FORT=1 TO 50:NEXT
```

Los sonidos más excitantes son los que únicamente puede producir el sintetizador de sonido, sin imitar a ningún instrumento conocido. Por ejemplo, pruebe:

```
20POKES+5,144:POKES+6,243:REM A=9;D=0;S=15;R=3
```


FILTRADO

El contenido en armónicos de una forma de onda puede ser cambiado usando un filtro. El chip SID está equipado con tres tipos de filtros. Pueden ser usados individualmente o en combinación con otros. Vuelva a cargar el programa de ejemplo número 1 para experimentar con los filtros. Hay varios controles de filtro que debe ajustar.

Añada la línea 15 al programa para ajustar la frecuencia de corte del filtro. La frecuencia de corte es el punto de referencia para el filtro. Usted debe ajustar el punto de corte para las frecuencias alta y baja en los registros 21 y 22. Para activar el filtro en la voz número 1, POKE en el registro 23.

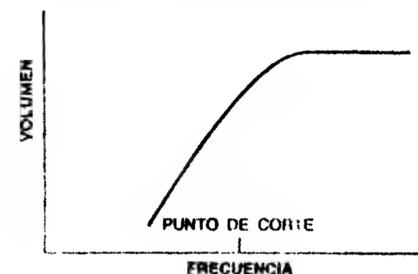
Después cambie la línea 30 para ver el uso del filtro pasa altos (Vea el mapa de registros del chip SID).

PROGRAMA DE EJEMPLO 5 (EJEMPLO 1 MODIFICADO):

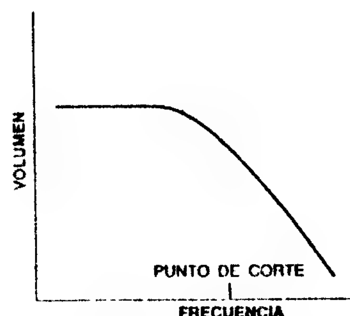
```
5 S=54272
10 FORL=STOS+24:POKEL,0:NEXT
15 POKES+22,128:POKES+21,0:POKES+23,1
20 POKES+5,9:POKES+6,0
30 POKES+24,79
40 READHF,LF,DR
50 IFHF<0THENEND
60 POKES+1,HF:POKES,LF
70 POKES+4,33
80 FORT=1TODR:NEXT
90 POKES+4,32:FORT=1TO50:NEXT
100 GOTO40
110 DATA25,177,250,28,214,250
120 DATA25,177,250,25,177,250
130 DATA25,177,125,28,214,125
140 DATA32,94,750,25,177,250
150 DATA28,214,250,19,63,250
160 DATA19,63,250,19,63,250
170 DATA21,154,63,24,63,63
180 DATA25,177,250,24,63,125
190 DATA19,63,250,-1,-1,-1
```

Ejecute (RUN) el programa. Advierta cómo los tonos bajos tienen su volumen cortado. La calidad de la nota es distinta. Esto sucede porque usa un filtro de pasa/alto que atenua (reduce el volumen) las frecuencias por debajo de la frecuencia de corte especificada.

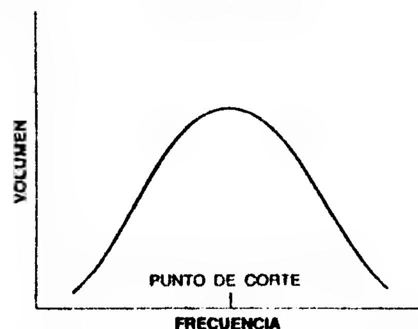
El chip SID de su Commodore posee tres tipos de filtros. Hemos usado el filtro pasaaltos. Este filtro deja pasar todas las frecuencias iguales o por encima del punto de corte, atenuando las situadas debajo del mismo.



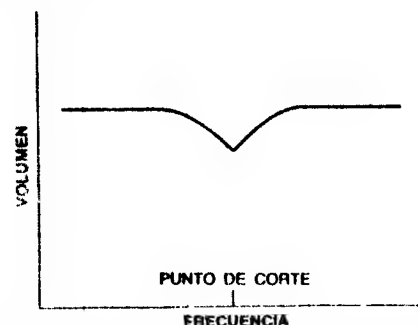
El chip SID dispone también de un filtro pasabajos. Como su nombre implica, este filtro deja pasar las frecuencias por debajo del punto de corte y atenua las que están por encima del mismo.



Finalmente, el chip está equipado con un filtro pasabanda, que deja pasar una banda limitada de frecuencias alrededor del punto de corte, atenuando las demás.



Los filtros pasaaltos y pasabajos se pueden combinar para formar un filtro que atenue las frecuencias de corte dejando pasar todas las demás.



El registro 24 determina el tipo de filtro que desee usar. Otra función del registro 24 es el control de volumen. El bit 6 controla el filtro pasaaltos (0=desactivado, 1=activado), el bit 5 es el que controla el filtro pasabanda y el bit 4 controla el filtro pasabajos. Los tres bits menos significativos de la frecuencia de corte se determinan por el registro 21 (de 0 a 7), mientras que los 8 bits de la alta frecuencia de corte se determinan por el registro 22 (de 0 a 255).

Actuando cuidadosamente con los filtros, usted puede cambiar la estructura de armónicos de cualquier forma de onda para crear el sonido que necesite. Los filtros combinados con las fases ADSR de los sonidos pueden producir efectos muy interesantes.

TECNICAS AVANZADAS

Los parámetros del chip SID pueden ser cambiados dinámicamente durante la ejecución de una nota para crear interesantes y divertidos efectos. Para poder realizar estos cambios fácilmente, salidas digitizadas del tercer oscilador y del tercer generador de envolvente están disponibles para usted en los registros 27 y 28, respectivamente.

La salida del oscilador 3 (registro 27) está directamente relacionada con la forma de onda seleccionada. Si selecciona la onda diente de sierra para el oscilador 3, el registro presenta una serie de números incrementados paso a paso de 0 a 255 en el tiempo determinado por la frecuencia del oscilador 3. Si selecciona la onda triangular, la salida se incrementa de 0 a 255 para después decrementar otra vez hasta 0. Si selecciona la onda de pulso, la salida salta de 0 a 255 y de 255 a 0 repetidamente. Finalmente, seleccionando la onda de ruido, la salida muestra números al azar. Cuando el tercer oscilador se usa para modulación, usted normalmente NO desea oír su salida. Ajustando el bit 7 del registro 24 se desactiva la salida de audio de la voz 3. El registro 27 siempre refleja los cambios en la salida del oscilador y no es afectado de ningún modo por el generador de envolvente (ADSR).

El registro 25 le da acceso a la salida del generador de envolvente del oscilador 3. Funciona del mismo modo que la salida del oscilador 3. El oscilador debe ser activado para producir cualquier salida de este registro.

El VIBRATO (una rápida variación de frecuencia) se puede lograr añadiendo la salida del oscilador 3 a la frecuencia de otro oscilador. El programa de ejemplo 6 ilustra esta idea.

PROGRAMA DE EJEMPLO 6:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+3,8
40 POKES+5,41:POKES+6,89
50 POKES+14,117
60 POKES+18,16
70 POKES+24,143
80 READFR,DR
90 IFFR=0THENEND
100 POKES+1,65
110 FORT=1TODR*2
120 FQ=FR+PEEK(S+27)/2
130 HF=INT(FQ/256):LF=FQAND255
140 POKES+0,LF:POKES+1,HF
150 NEXT
160 POKES+4,64
170 GOTO80
500 DATA4817,2,5183,2.5407,2
510 DATA8583,4,5407,2,8583,4
520 DATA5407,4.8583.12.9634,2
540 DATA9634,4,10814,2,8583,2
550 DATA9634,4,8583,12
560 DATA0,0
```

He aquí la explicación línea a línea del programa de ejemplo 6.

EXPLICACION LINEA A LINEA DEL PROGRAMA DE EJEMPLO 6.

Línea (s)	Descripción
10	Asigna la variable S el inicio del chip SID.
20	Borra los registros del chip de sonido.
30	Coloca la anchura de pulso para la voz 1.
40	Ajusta el Ataque/Decaimiento en la voz 1 (A=2, D=9). Ajusta Sostenimiento/Relajación voz 1 (S=5, R=9).
50	Ajusta baja frecuencia para la voz 3.
60	Coloca la forma de onda triangular en la voz 3.
70	Coloca el volumen a 15, desactiva la salida de audio de la voz 3.
80	Lee la frecuencia y duración de la nota.
90	Si la frecuencia es igual a 0, el programa para.
100	POKE el inicio del control de forma de onda voz 1.
110	Inicio del bucle de duración de la nota.
120	Calcula nueva frecuencia utilizando la salida del tercer oscilador.
130	Calcula las frecuencias alta y baja.
140	POKE alta y baja frecuencia en la voz 1.
150	Fin del bucle de duración de la nota.
160	POKE paro de control de la forma de onda voz 1.
170	Regresa para procesar la siguiente nota.
500–550	Frecuencias y duraciones para la canción.
560	Ceros señalando el fin de la canción.

Una gran variedad de efectos de sonido pueden ser conseguidos usando efectos dinámicos. Por ejemplo, el programa del sonido de una sirena mostrado a continuación cambia dinámicamente la frecuencia de salida del oscilador 1 basándose en la salida de la onda triangular del oscilador número 3.

PROGRAMA DE EJEMPLO 7:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+14,5
40 POKES+18,16
50 POKES+3,1
60 POKES+24,143
70 POKES+6,240
80 POKES+4,65
90 FR=5389
100 FORT=1TO200
110 FQ=FR+PEEK(S+27)*3.5
120 HF=INT(FQ/256):LF=FQ-HF*256
130 POKES+0,LF:POKES+1,HF
140 NEXT
150 POKES+24,0
```


EXPLICACION LINEA A LINEA DEL PROGRAMA DE EJEMPLO 7:

Línea (s)	Descripción
10	Asigna a la variable S el inicio del chip SID.
20	Borra los registros de sonido del chip.
30	Ajusta baja frecuencia en voz 3.
40	Coloca onda triangular en voz 3.
50	Ajusta anchura de pulso en voz 1.
60	Coloca el volumen a 15, desactiva la salida de audio de la voz 3.
70	Ajusta Sostenimiento/Relajación voz 1 (S=15, R=0).
80	POKE inicio control de forma de onda voz 1.
90	90 Ajusta la baja frecuencia para la sirena.
100	Inicia el bucle de tiempo.
110	Calcula nueva frecuencia usando la salida del oscilador número 3.
120	Calcula la alta y baja frecuencia.
130	POKE alta y baja frecuencia en voz 1.
140	Fin del bucle de tiempo.
150	Desactiva el volumen.

La forma de onda de ruido blanco se puede usar para crear una gran cantidad de efectos de sonido. El siguiente programa imita aplausos usando la forma de onda de ruido filtrada:

PROGRAMA DE EJEMPLO 8:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+0,240:POKES+1,33
40 POKES+5,8
50 POKES+22,104
60 POKES+23,1
70 POKES+24,79
80 FORN=1TO15
90 POKES+4,129
100 FORT=1TO250:NEXT:POKES+4,128
110 FORT=1TO30:NEXT:NEXT
120 POKES+24,0
```

EXPLICACION LINEA A LINEA DEL PROGRAMA DE EJEMPLO 8:

Línea (s)	Descripción
10	Asigna la variable S el inicio del chip de sonido.
20	Borra los registros del chip de sonido.
30	Ajusta las frecuencias alta y baja para la voz 1.
40	Ajusta Ataque/Decaimiento voz 1 (A=0, D=8).
50	Ajusta la alta frecuencia de corte del filtro.
60	Activa el filtro para la voz 1.
70	Volumen a 15, filtro pasaaltos.
80	Cuenta 15 palmadas.
90	Inicio del control de forma de onda.
100	Espera. (Bucle vacío). Después para el control de forma de onda.
110	Espera. Después pasa a la siguiente palmada.
120	Desactiva el volumen.

SINCRONIZACION Y MODULACION DE TIMBRE

El chip 6581 (SID) le permite crear estructuras de armónicos más complejas mediante la sincronización y modulación de timbre de dos voces.
El proceso de sincronización es básicamente una operación lógica AND de dos formas de onda. Cuando una u otra es cero, la salida es cero. El siguiente ejemplo utiliza esta técnica para imitar a un mosquito:

PROGRAMA DE EJEMPLO 9:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+1,100
40 POKES+5,219
50 POKES+15,28
60 POKES+24,15
70 POKES+4,19
80 FORT=1TO5000:NEXT
90 POKES+4,18
100 FORT=1TO1000:NEXT:POKES+24,0
```


EXPLICACION LINEA A LINEA DEL PROGRAMA DE EJEMPLO 9:

Línea (s)	Descripción
10	Asigna la variable S el inicio del chip de sonido.
20	Borra los registros del chip de sonido.
30	Ajusta la alta frecuencia para la voz 1.
40	Ajusta Ataque/Decaimiento para la voz 1.
50	Ajusta la alta frecuencia para la voz 3.
60	Coloca el volumen a 15.
70	Ajusta el inicio de onda triangular, sincroniza el control de forma de onda para la voz 1.
80	Bucle de tiempo.
90	Ajusta el paro de la onda triangular, sincroniza el control de forma de onda para la voz 1.
100	Espera. Después, desactiva el volumen.

La sincronización se activa en la línea 70 mediante el ajuste de los bits 0,1 y 4 del registro 4. El bit 1 activa la sincronización entre las voces 1 y 3. Los bits 0 y 4 tienen las funciones usuales de activar la voz 1 y ajustar la forma de onda triangular.
La modulación de timbre (activada en la voz 1 ajustando el bit 3 del registro 4 en la línea 70 del siguiente programa) reemplaza la salida triangular del oscilador 1 por una combinación con el "timbre modulado" de los osciladores 1 y 3. Esto produce una estructura de sobretono no armónico que sirve para imitar el sonido de campanas y gongs. El siguiente programa reproduce las campanadas de un reloj:

PROGRAMA DE EJEMPLO 10:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+1,130
40 POKES+5,9
50 POKES+15,30
60 POKES+24,15
70 FORL=1TO12:POKES+4,21
80 FORT=1TO1000:NEXT:POKES+4,20
90 FORT=1TO1000:NEXT:NEXT
```

EXPLICACION LINEA A LINEA DEL PROGRAMA DE EJEMPLO 10:

Línea (s)	Descripción
10	Asigna a la variable S el inicio del chip de sonido.
20	Borra los registros del chip de sonido.
30	Ajusta la alta frecuencia para la voz 1.
40	Ajusta el Ataque/Decaimiento voz 1. (A=0, D=9).
50	Ajusta la alta frecuencia para la voz 3.
60	Ajusta el volumen a 15.
70	Cuenta las campanadas, ajusta el inicio de la onda triangular, control de modulación de timbre de la voz 1.
80	Bucle de tiempo, para la onda triangular, modulación de timbre.
90	Bucle de tiempo, próxima campanada.

Los efectos disponibles mediante el uso de los Registros de Control del chip SID son numerosos y variados. El único modo de conocer y apreciar todas las capacidades de su Commodore 64 es practicar y experimentar. Los ejemplos contenidos en esta sección del Manual de Referencia del Programador únicamente exploran una pequeña parte de las posibilidades del chip SID.
Pronto aparecerá el libro **CREANDO MUSICA CON SU COMMODORE 64** que contiene desde los más simples divertimentos y juegos hasta la instrucción musical de tipo profesional.

CAPITULO 5

DE BASIC A LENGUAJE MAQUINA

- ¿Qué es el Lenguaje Máquina?
- ¿Cómo puede usted escribir programas en Lenguaje Máquina?
- Notación Hexadecimal
- Modos de direccionamiento
- Indexación
- Subrutinas
- Notas útiles para el principiante
- Acercamiento a una tarea importante
- Juego de instrucciones del microprocesador MCS6510
- Organización de la memoria del Commodore 64
- El KERNAL
- Actividades del KERNAL durante la puesta en marcha
- Uso del Lenguaje Máquina desde el BASIC
- Mapa de memoria del Commodore 64

QUE ES EL LENGUAJE MAQUINA?

El corazón de todos los microordenadores es el microprocesador central. Es un chip muy especial que actúa como el "cerebro" del ordenador. El Commodore 64 no es una excepción. Cada microprocesador entiende un conjunto de instrucciones (o lenguaje) propio. Estas instrucciones se llaman instrucciones del lenguaje máquina. Para ser más precisos, el lenguaje máquina es el UNICO lenguaje de programación que comprende el microprocesador. Es el lenguaje NATIVO de la máquina.

Si el lenguaje máquina es el único lenguaje que comprende el Commodore 64, entonces, ¿cómo comprende también el CBM BASIC? El CBM BASIC NO ES el lenguaje máquina del Commodore 64. Pero, ¿cómo el Commodore 64 entiende instrucciones en BASIC como PRINT o GOTO?

Para contestar a esta pregunta, primero debe saber lo que sucede en el interior de su Commodore 64. Además del microprocesador, que es el cerebro de su ordenador, existe también un programa en lenguaje máquina almacenado en una memoria especial que no puede ser modificada. Y, aún más importante, este programa no se borra al desconectar el ordenador, al revés de lo que ocurre con un programa escrito por usted. Este programa en lenguaje máquina se llama el SISTEMA OPERATIVO del Commodore 64. Su ordenador sabe lo que debe hacer gracias a que cuando se conecta, este programa se ejecuta automáticamente. El SISTEMA OPERATIVO se encarga de "organizar" la memoria de su máquina para que realice distintas tareas. También comprueba los caracteres que escribe desde el teclado y los coloca en la pantalla, además de realizar muchas otras funciones. El SISTEMA OPERATIVO puede ser considerado como la "inteligencia y personalidad" de su ordenador. De forma que cuando conecta el ordenador, el SISTEMA OPERATIVO toma el control de la máquina y –una vez realizadas las tareas de inicialización– le dice:

READY (listo)

El SISTEMA OPERATIVO del Commodore 64 le permite a partir de ahora escribir en el teclado y utilizar el EDITOR DE PANTALLA incorporado a su ordenador. El EDITOR DE PANTALLA le permite mover el cursor, insertar y borrar letras, etc., y, de hecho, es sólo una parte del sistema operativo que se ha incorporado para una mayor comodidad en la entrada y edición de programas.

Todas las instrucciones y comandos del BASIC son reconocidas por otro programa en lenguaje máquina presente en su Commodore 64. Este programa ejecuta la parte correspondiente del mismo que corresponde a las instrucciones del CBM BASIC que ha escrito usted. Este programa se llama INTERPRETE BASIC, ya que cada instrucción es interpretada por orden, una a una por él mismo, a menos que encuentre alguna instrucción que no comprenda, en cuyo caso nos mostrará el conocido mensaje:

?SYNTAX ERROR

READY

QUE ASPECTO TIENE EL CODIGO MAQUINA?

Usted estará probablemente familiarizado con los comandos BASIC PEEK y POKE, que le permiten leer y cambiar los valores de las posiciones de memoria.

Probablemente habrá usado estas instrucciones para dibujar gráficos en la pantalla o crear efectos de sonido.

Cada posición de memoria posee un número que la identifica. Este número es conocido como la "dirección" de memoria. Si imagina la memoria del Commodore 64 como una calle llena de edificios, el número de cada puerta será, por supuesto, la dirección. Ahora veamos las distintas secciones de la "calle" y sus propósitos:

MAPA DE MEMORIA SIMPLIFICADO DEL COMMODORE 64

DIRECCION	DESCRIPCION
0 y 1	–Registros del 6510
2 hasta 1023	–Inicio de la memoria –Memoria usada por el Sistema Operativo
1024 hasta 2039	–Memoria de pantalla
2040 hasta 2047	–Punteros de los SPRITES
2048 hasta 40959	–Esta es SU memoria. Aquí se pueden almacenar programas en BASIC, Lenguaje Máquina o ambos
40960 hasta 49151	–8K del intérprete CBM BASIC
49152 hasta 53247	–RAM para programas especiales
53248 hasta 53294	–Registros del VIC-II
55296 hasta 56296	–RAM de color
56320 hasta 57343	–Registros de Entradas/Salidas
57344 hasta 65535	–8K del Sistema Operativo KERNAL CBM

Si no comprende lo que significa cada parte de la memoria, en otras partes de este manual encontrará explicaciones detalladas.
Los programas en Lenguaje Máquina (C/M) consisten en instrucciones que pueden o no tener operandos (parámetros) asociados con ellas. Cada instrucción ocupa una posición de memoria, y cada operando se almacena en una o dos posiciones siguiendo a la instrucción.
En sus programas en BASIC, las palabras como PRINT o GOTO ocupan, de hecho, una sola posición de memoria, en lugar de una por cada letra de la palabra. El contenido de la posición que representa una palabra reservada BASIC determinada se llama "token" (señal). En lenguaje máquina, hay distintas señales para las distintas instrucciones, pero también ocupan un solo byte (byte= posición de memoria).
Las instrucciones en lenguaje máquina son muy simples. Por esto, una instrucción no puede realizar grandes tareas. Las instrucciones en lenguaje máquina pueden cambiar el contenido de una posición de memoria o alguno de los registros internos (posiciones especiales de almacenamiento) del microprocesador. Los registros internos son la base del lenguaje máquina.

LOS REGISTROS DEL MICROPROCESADOR 6510

EL ACUMULADOR

Este es el registro MAS IMPORTANTE del microprocesador. Hay varias instrucciones en C/M que le permiten copiar el contenido de una posición de memoria en el acumulador, copiar el contenido del acumulador en una posición de memoria, modificar directamente el contenido del acumulador u otro registro, sin afectar a la memoria. Y además el acumulador es el único registro que dispone de instrucciones para realizar operaciones matemáticas.

EL REGISTRO X

Este es un registro importante. Dispone de instrucciones para realizar todas las transformaciones que se pueden realizar con el acumulador. Pero hay otras instrucciones para realizar cosas que sólo puede hacer el registro X. Varias instrucciones en lenguaje máquina le permiten modificar el contenido del registro X sin afectar a la memoria, copiar el contenido de una posición de memoria en este registro, o copiar el contenido del registro en una posición de memoria.

EL REGISTRO Y

Este es otro importante registro. Hay instrucciones para realizar con él todas las transformaciones realizables con el acumulador y el registro x. Pero hay también otras instrucciones que sirven para lograr cosas que sólo puede realizar el registro Y. Varias instrucciones en lenguaje máquina le permiten copiar el contenido de una posición de memoria en el registro Y, copiar el contenido de este registro en una posición de memoria y modificar este contenido sin afectar a la memoria.

EL REGISTRO DE ESTADO

Este registro consiste en 8 "banderas" (flags) (una bandera=cualquier señal que indica que alguna cosa ha, o no ha ocurrido).

EL CONTADOR DE PROGRAMA

Este registro contiene la dirección de la instrucción en C/M que se está ejecutando en cada momento. Puesto que el sistema operativo está "ejecutándose" (RUN) permanentemente en su Commodore 64 (Como en todos los ordenadores), el contador de programa está cambiando siempre. Sólo puede ser parado deteniendo de alguna forma el microprocesador.

EL PUNTERO DE STACK (STACK= PILA)

Este registro contiene la posición de la primera posición disponible en el stack. El stack se usa como almacén temporal de datos por un programa en lenguaje máquina y por el ordenador.

EL PORT DE E/S

Este registro se encuentra en las posiciones 0 (para el REGISTRO DE DIRECCION DE DATOS) y 1 (para el actual PORT). Es un port de E/S de 8 bits. En el Commodore 64, se usa este registro para el manejo de la memoria, permitiendo al microprocesador controlar más de 64K de memoria RAM y ROM.
Los detalles de estos registros no se encuentran aquí. Se han descrito someramente para que comprenda los principios que se explicarán a continuación.

¿COMO PUEDE USTED ESCRIBIR PROGRAMAS EN LENGUAJE MAQUINA?

Puesto que los programas en C/M residen en memoria, y el Commodore 64 no posee características que le permitan escribir y depurar con facilidad estos lenguajes, deberá usar un programa que le de estas facilidades, o escribir usted mismo un programa en BASIC que le "permita" escribir programas en lenguaje máquina.
El método más común de escribir un programa en C/M es mediante unos programas llamados ensambladores. Estos programas le permiten escribir las instrucciones de C/M en formato mnemónico, lo que hace que un programa escrito de esta forma sea más legible que una lista de números. Recuerde: Un programa que le permite escribir programas en C/M en formato mnemónico se llama ENSAMBLADOR. Lógicamente, un programa que se encargé de mostrar en formato mnemónico un programa en C/M residente en la memoria del ordenador es llamado DESENSAMBLADOR. Uno de los cartuchos fabricados por Commodore para su C-64 es el llamado MONITOR DE LENGUAJE MAQUINA que contiene ensamblador, desensamblador, etc.

64MON

El cartucho 64MON, disponible en su tienda habitual, es un programa que le permite viajar del mundo del BASIC al del lenguaje máquina. Puede mostrar el contenido de

los registros internos del microprocesador, le permite también ver áreas de memoria y modificarlas en la pantalla, usando el editor de pantalla incorporado. Posee también un ensamblador/desensamblador, así como otras características que le permitirán programar fácilmente en lenguaje máquina. El uso de un ensamblador NO ES imprescindible para escribir programas en C/M, pero le permite hacerlo de forma mucho más fácil. Si desea escribir frecuentemente en C/M deberá pensar en la compra de un ensamblador. Sin éste deberá con toda probabilidad entrar sus programas mediante POKE, lo que –ademas de muy engorroso– es sumamente impracticable por la facilidad con que se cometen errores. Los ejemplos mostrados en este capítulo están escritos en el formato del 64MON. Sin embargo, todos los ensambladores tienen un formato parecido, por lo que dichos ejemplos serán compatibles con cualquiera de ellos.
Antes de explicar otras características del 64MON debemos estudiar el sistema de numeración hexadecimal.

NOTACION HEXADECIMAL

La mayoría de programadores en lenguaje máquina utilizan la notación hexadecimal para referirse a las distintas posiciones de memoria de sus programas. Algunos ensambladores le permiten referirse a direcciones y números en decimal (base 10), binario (base 2), octal (base 8) y, por supuesto, en hexadecimal (base 16) (o "hex", como dice la mayoría de la gente). Estos ensambladores realizan las conversiones por usted.

DECIMAL	HEXADECIMAL	BINARIO
0	0	00000000
1	1	00000001
2	2	00000010
3	3	00000011
4	4	00000100
5	5	00000101
6	6	00000110
7	7	00000111
8	8	00001000
9	9	00001001
10	A	00001010
11	B	00001011
12	C	00001100
13	D	00001101
14	E	00001110
15	F	00001111
16	10	00010000

La notación hexadecimal puede parecerle al principio un poco engorrosa o difícil de aprender, pero como en la mayoría de las cosas, no tardará en tener una gran soltura si practica un poco.
Estudiando los números decimales (base 10), se dará cuenta de que cada dígito tiene un rango entre 0 y un número igual a su base menos 1 (en la notación decimal (base 10), base -1 = 9, es decir, el rango va de 0 a 9). ESTA REGLA SE APLICA A TODAS LAS BASES. Los números binarios (base 2) tienen dígitos que van de 0 a 1 (1=base-1). De forma similar, los dígitos de los números hexadecimales van de 0 a 15 pero puesto que no dispoemos de dígitos únicos que representen un valor superior a 9, se han usado las 6 primeras letras del alfabeto.

Veámoslo de otra manera; he aquí un ejemplo de cómo se construye un número en base 10 (decimal):

Base elevada a	10^3	10^2	10^1	10^0
potencias crecientes.....	1000	100	10	1
Equivale a	4	5	6	9

Considere 4569 (base 10)
=(4X1000)+(5X100)+(6X10)+9

Ahora vea un ejemplo de cómo se construye un número en base 16 (número hexadecimal):

Base elevada a	16^3	16^2	16^1	16^0
potencias crecientes.....	4096	256	16	1
Equivale a..............	1	1	D	9

Considere 11D9 (base 16)
=1X4096+1X256+13X16+9

Por esto, 4569 (base 10)=11D9 (base 16).
Como se explicó antes, se pueden direccionar 65536 posiciones de memoria (de 0 a 65535). Este rango en hexadecimal va de 0 a FFFF.
Normalmente los números hexadecimales se escriben precedidos del signo dólar ($) De esta forma se puede distinguir un número hexadecimal de uno decimal. Ahora veamos algunos números hexadecimales usando el 64MON. Escriba:

B

```
   PC   SR   AC   XR   YR   SP
.; 0401 32   04   5E   00   F6
```
(Pueden ser distintos)

Ahora si escribe:
.M 0000 0020 (y pulse **RETURN**)

usted verá filas de 6 numeros hexadecimales. El primer numero –de 4 digitos– es la direccion de memoria del primer byte de cada fila, y los otros cinco números mues-

tran el contenido de las posiciones de memoria a partir de la dirección inicial de la línea.
Usted debe intentar "pensar" en hexadecimal. No es muy difícil, porque no tiene que convertir los números a decimal.
Por ejemplo, si usted dice que se almacena un determinado valor en la posición $14ED en lugar de 5357, no hay ninguna diferencia.

SU PRIMERA INSTRUCCION EN LENGUAJE MAQUINA

LDA–CARGA EL ACUMULADOR

En el lenguaje ensamblador del 6510, los mnemónicos tienen siempre 3 caracteres. LDA representa "carga el acumulador con..." y lo que debe cargarse en el acumulador se determina por los parámetros asociados a la instrucción. El ensamblador conoce las señales de cada instrucción, y lo único que hace cuando "ensambla" una instrucción es colocar en memoria (en la posición especificada por usted) la señal (token) de la instrucción seguida de los parámetros que haya entrado. Algunos ensambladores producen mensajes de error, o avisan cuando intenta ensamblar algo que el programa o el microprocesador no conocen.
Si usted coloca el símbolo "#" delante de los parámetros asociados con la instrucción, significa que desea cargar el registro afectado con el valor que sigue al símbolo "#". Por ejemplo:

```
LDA # $05          $=HEX
```

Esta instrucción coloca el valor $05 (decimal 5) en el acumulador. El ensamblador coloca en la dirección de memoria deseada para esta instrucción $A9 (que es la señal para esta instrucción particular, en este modo), y coloca $05 en la posición de memoria que sigue a la que almacena el código de la instrucción ($A9).
Si el parámetro asociado a una instrucción tiene delante el símbolo "#", el parámetro es un "valor", en lugar de el contenido de una posición de memoria u otro registro. Esto se llama modo "inmediato". Para comprenderlo mejor, compare con otro modo:
Si desea colocar el contenido de la posición de memoria e$102E en el acumulador, estará usando el modo "absoluto" de la instrucción:

```
LDA $102E
```

El ensamblador puede distinguir entre los dos modos porque el último no tiene un "#" delante del parámetro. El microprocesador 6510 distingue los dos modos porque la instrucción posee un código distinto para cada modo. LDA (inmediato) posee el código $A9, y LDA (absoluto) posee el código $AD.
El mnemónico que representa cada instrucción indica normalmente su uso. Por ejemplo, consideremos la instrucción LDX: ¿que cree usted que realiza?
Si ha respondido "carga el registro X con..." enhorabuena, paso al final de la lección. Si no ha sido así no se preocupe, el lenguaje máquina requiere paciencia y no se puede aprender en un día.
Los varios registros internos pueden ser considerados como posiciones de memoria, ya que pueden albergar un byte de información. No es necesario explicar el sistema de numeración binario (base 2) puesto que sigue las mismas reglas explicadas para los otros sistemas, pero sepa que un "bit" es un dígito binario y un "byte" es un conjunto de 8 bits. Esto significa que el máximo valor que puede tener un byte es el máximo valor que se puede obtener con un número binario de 8 dígitos. Este número es 11111111 (binario), que equivale a $FF (hex) y a 255 (decimal). Probablemente habrá comprendido que sólo se pueden colocar números entre 0 y 255 en una posición de memoria. Si intenta POKE 7680,260 (que es la instrucción BASIC que "dice": "Coloca en la posición de memoria 7680 el valor 260"), se encontrará conque el intérprete BASIC sabe que una posición de memoria sólo puede contener números entre 0 y 255, por lo que su Commodore 64 replicará:


```
?ILLEGAL QUANTITY ERROR

READY
■
```

Si el límite de contenido de un byte es $FF (hex), ¿cómo se expresa en memoria el parámetro de la instrucción absoluta "LDA $102E"?. Es expresada en dos bytes. (Por supuesto, ya que no cabe en uno). Los dos dígitos bajos (de la derecha) del número hex forman el "byte bajo" de la dirección, y los dos dígitos altos (de la izquierda) del número hex forman el "byte alto" de la dirección.
El 6510 requiere que cualquier dirección se especifique con el byte bajo primero, y a continuación el byte alto. Esto significa que la instrucción "LDA $102E" se expresa en memoria como 3 valores (bytes) consecutivos:

```
$AD,$2E,$10
```

Ahora sólo necesita conocer otra instrucción para poder escribir su primer programa. Esta instrucción es BRK. Para una completa explicación de ésta y el resto de instrucciones de lenguaje maquina refiérase al libro M.O.S. 6502 Programming Manual. Pero, por ahora, piense que la instrucción BRK es muy similar a la END del BASIC.
Si escribe un programa con el 64MON y coloca la instrucción BRK al final, cuando se ejecuta el programa se vuelve al 64MON al encontrarla. Esto no sucedera si hay algún error en su programa, o si no se encuentra la instrucción BRK (al contrario que en BASIC, en que no es necesario colocar END para que se termine un programa). Por supuesto, la tecla STOP le permitirá parar el programa.

ESCRIBIENDO SU PRIMER PROGRAMA

Si ha usado POKE para colocar caracteres en la pantalla se habrá dado cuenta de que los códigos para POKE difieren de los códigos CBM ASCII. Por ejemplo, s[illegible]tra:

```
PRINT ASC("A") (y pulsa RETURN)
```

El Commodore 64 responde:

```
65

READY
```

Sin embargo, para colocar una "A" en la pantalla mediante POKE, el código es 1. Entre:

SHIFT CLR/HOME para borrar la pantalla

POKE 1024,1 (y RETURN) (1024 es el inicio de la memoria de pantalla)

La "P" de la instrucción POKE se convertirá en una "A".
Ahora pruebe lo mismo en lenguaje máquina. Escriba lo siguiente en el 64MON: (El cursor debe estar parpadeando a la derecha de un punto (.)).

.A 1400 LDA #$01 (y pulse RETURN)

El Commodore 64 le responde:

```
.A 1400 LDA #$01
.A 1402
```

Escriba: STA $0400

(La instrucción STA coloca el contenido del acumulador en una posición de memoria especificada).
El Commodore 64 muestra: .A 1405

Ahora escriba: BRK

Limpie la pantalla y escriba: G 1400

La "G" se convertirá en una "A" si ha realizado correctamente todas las operaciones.
Ahora ya ha escrito su primer programa en lenguaje máquina. Su propósito es colocar la letra A en la primera posición de la memoria de pantalla (esquina superior izquierda). Habiendo aprendido esto, ahora podemos explorar otras instrucciones y principios.

MODOS DE DIRECCIONAMIENTO

PAGINA CERO

Como se indicó antes, las direcciones absolutas se expresan en términos de byte bajo y byte alto. El byte alto con frecuencia se refiere a la PAGINA de memoria. Por ejemplo, la dirección $1637 se encuentra en la página $16 (22) y $0277 se encuentra en la página $02 (2). Existe sin embargo un modo especial de direccionamiento que se conoce como direccionamiento de página cero y, como su nombre indica, está asociado con el direccionamiento de las posiciones de memoria contenidas en dicha página. Cuando se utiliza este modo de direccionamiento, se asume que la dirección tiene SIEMPRE el byte alto con valor cero. El modo de direccionamiento de página cero permite expresar una dirección con sólo un byte (el byte bajo) en lugar de los dos que se utilizan en el direccionamiento absoluto. Como hemos dicho antes, en este modo el microprocesador asume que el byte alto tiene el valor de cero. Por esto, el direccionamiento de página cero puede referirse exclusivamente a posiciones entre $0000 y $00FF. Esta información y modo de direccionamiento puede que no le parezcan importantes, pero más adelante necesitará recordar los principios del direccionamiento de página cero.

LA PILA (EL STACK)

El 6510 posee algo conocido como stack (pila). Se usa por el programador y el microprocesador para almacenar datos temporales y recordar, por ejemplo, el orden en que han ocurrido diversas cosas. La instrucción del BASIC GOSUB, que permite al programa ejecutar una subrutina, debe recordar el punto desde el que ha sido llamada, de forma que cuando el intérprete BASIC encuentre la instrucción RETURN sepa a donde ha de dirigirse. Cuando en un programa en BASIC el intérprete ejecuta una instrucción GOSUB, se "coloca" la posición actual del programa en la pila antes de ejecutar la subrutina, y cuando se ejecuta RETURN, el intérprete "saca" la información de la pila para devolver la ejecución del programa al punto adecuado. El intérprete usa instrucciones como **PHA**, que coloca en la pila el contenido del acumulador, y **PLA** (la inversa) que extrae un valor de la pila y lo coloca en el acumulador. El registro de estado se puede colocar y extraer de la pila mediante **PHP** y **PLP** respectivamente.
La pila es una zona de memoria de 256 bytes de largo, colocada en la página 1 de la memoria. Por esto ocupa las posiciones $0100 a $01FF. Esta zona de memoria está organizada al revés. En otras palabras, la primera posición de la pila es $01FF y la última $0100. Otro registro del procesador es el llamado puntero de pila (stack pointer), que apunta siempre a la primera posición disponible en la pila. Cuando se coloca algo en la pila, se coloca siempre en la posición contenida en el puntero, y dicho puntero se mueve a la siguiente posición (decrementado). Cuando algo se extrae de la pila, se incrementa el puntero de pila, y el byte al que apunta el mismo es colocado en el registro especificado.
Hasta este punto hemos cubierto las instrucciones en modo inmediato, de página cero y absoluto. También hemos cubierto –aunque realmente no hayamos hablado de él– el modo "implícito". El modo implícito significa que la información está contenida implícitamente en la propia instrucción. En otras palabras, la instrucción ya indica qué registros, banderas y memorias se utilizan. Los ejemplos son PHA, PLA, PHP, y PLP, que se refieren al proceso en la pila, y al acumulador y el registro de estado, respectivamente.

NOTA: A partir de este punto se utilizarán las siguientes abreviaciones: X por registro X; Y por registro Y; A por acumulador; S por puntero de pila y P por estado del procesador.

INDEXACION

La indexación es una parte muy importante en el lenguaje máquina del 6510. Puede definirse como "crear la dirección actual mediante la suma de la dirección de base y el contenido de los registros X o Y."
Por ejemplo, si X contiene $05, y el microprocesador ejecuta la instrucción LDA en el "modo absoluto indexado X" con la dirección de base $9000, la dirección cuyo

contenido se cargará en el acumulador será $9000 + $05 = $9005. El formato mnemónico de una instrucción en modo absoluto indexado es el mismo que para el modo absoluto excepto que a la dirección se añade "X" o "Y" para indicar que se suma el índice a la dirección:

EJEMPLO:

LDA $9000, X

En el microprocesador 6510 están disponibles los modos absoluto indexado, indexado de página cero, indexado indirecto e indirecto indexado.

INDEXADO INDIRECTO

Este modo le permite únicamente utilizar el registro Y como índice. La dirección actual debe ser de página cero, y el modo de la instrucción se llama indirecto porque la dirección especificada en la instrucción contiene el byte bajo de la dirección actual, y el siguiente byte contiene el byte alto de la dirección actual.

EJEMPLO:

Suponga que la dirección $01 contiene $45, y que la dirección $02 contiene $1E. Si la instrucción para cargar el acumulador en modo indexado indirecto se ejecuta y la posición de memoria de página cero debe ser $01, entonces la dirección actual debe ser:

Orden bajo = contenido de $01
Orden alto = contenido de $02
Registro Y = $00

Por lo que la dirección actual es igual a $1E45 + Y = $1E45
El nombre de este modo implica de hecho un principio de "indirección" que puede ser difícil de comprender en un primer momento: Veámoslo de otro modo:

"Voy a depositar esta carta en la oficina de correos situada en la CALLE MEMORIA $01, y la dirección de la carta es $05 casas después de la CALLE MEMORIA $1600." A continuación se muestra el código equivalente:

```
LDA #$00      - Carga el byte bajo de la dirección de base.
STA #$02      - Ajusta el byte bajo de la dirección indirecta.
LDA #$16      - Carga el byte alto de la dirección indirecta.
STA #$03      - Ajusta el byte alto de la dirección indirecta.
LDY #$05      - Ajusta el índice indirecto (Y).
LDA ($02),Y   - Carga indexando indirectamente con Y.
```

INDIRECTO INDEXADO

El indirecto indexado sólo le permite el uso del registro X como índice. Es parecido al anterior, pero es la dirección contenida en el puntero de página cero la que es indexada, en lugar de hacerlo con la dirección de base. Por esto, la dirección de base ES realmente la dirección de base puesto que el índice X ya ha sido usado para la indirección. El indirecto indexado se puede usar también si tenemos una tabla de punteros indirectos colocada en la página cero, puesto que el registro X puede entonces especificar el puntero a usar.

EJEMPLO:

Suponga que la posición $02 contiene $45, y la posición $03 contiene $10. Si la instrucción para cargar el acumulador en el modo indirecto indexado se ejecuta y la dirección especificada de página cero debe ser $02, la dirección actual será:

Orden bajo = contenido de ($02+X)
Orden alto = contenido de ($03+X)
registro X = $00

Por lo que el puntero actual es igual a $02 + X = $02.
La dirección actual es la dirección indirecta contenida en $02 y $03, o sea $1045.
Esto puede ser difícil de comprender. Veámoslo de otro modo:
"Voy a llevar esta carta a la quinta oficina de correos a partir de la CALLE MEMORIA $01, y la dirección de la carta es CALLE MEMORIA $1600." Este es el código equivalente:

```
LDA #$00      - Carga el byte bajo de la dirección de base.
STA #$06      - Ajusta el byte bajo de la dirección indirecta.
LDA #$16      - Carga el byte alto de la dirección de base.
STA #$07      - Ajusta el byte bajo de la dirección indirecta.
LDX #$05      - Ajusta el índice indirecto (X).
LDA ($01,X)   - Carga indexando indirectamente por X.
```

NOTA: De los dos modos de direccionamiento indirecto el primero es mucho más usado que el segundo.

RAMIFICACIONES Y TESTS

Otro principio muy importante del lenguaje máquina es su habilidad para comparar y detectar ciertas condiciones, de forma similar a la estructura "IF...THEN, IF...GOTO" en CBM BASIC.
Hay varias banderas en el registro de estado que son afectadas por distintas instrucciones de diferente forma. Por ejemplo, hay una bandera que se activa cuando una instrucción ha causado un resultado cero, y se desactiva cuando el resultado no es cero. La instrucción:

LDA A$00

causa la activación de la bandera de resultado cero, puesto que el resultado de la misma es que el acumulador contenga cero.

Hay un juego de instrucciones que permiten –cuando se da una condición particular– saltar a otra parte del programa. Un ejemplo de estas instrucciones es BEQ, que significa: salta si el resultado es cero. Las instrucciones de ramificación saltan si la condición es cierta y, si no lo es, el programa continúa en la siguiente instrucción, como si no hubiera ocurrido nada. Las instrucciones de ramificación no saltan por el resultado de la(s) anterior(es) instrucción(es), sino que examinan el registro de estado. Como se ha mencionado, existe en este registro una bandera de resultado cero. La instrucción BEQ salta si esta bandera (conocida como Z) está activada. Cada instrucción de ramificación tiene su opuesta. La instrucción BEQ tiene su opuesta en BNE, que significa salta si el resultado no es igual a cero. (Si Z no está activada).

Los registros de índice tienen un número de instrucciones asociadas que modifican su contenido. Por ejemplo, la instrucción INX INcrementa el registro X. Si el registro X contiene $FF (el máximo valor que puede tener X) antes de ser incrementado, dicho registro volverá a cero. Si desea que un programa haga algo hasta que el registro X contenga cero, puede usar la instrucción BNE para hacer un bucle que proseguirá hasta que se dé la condición mencionada (que X=0).

La instrucción opuesta a INX es DEX, que significa DEcrementa el registro X. Si el registro X contiene 0 y se ejecuta DEX, este registro pasará a tener el valor 255. Las instrucciones INY y DEY actúan de forma similar pero utilizando el registro Y. Pero qué se debe hacer si queremos que un programa no realice nada hasta que los registros X o Y tengan un valor determinado? Existen para este fin las instrucciones de comparación, CPX y CPY, que le permiten comparar los registros de índice con valores concretos, o incluso con el contenido de direcciones de memoria. Si desea ver si el registro X contiene $40, deberá usar la instrucción:

```
CPX #$40          – Compara X con el "valor" $40
BEQ               – Salta a otra parte del programa si la condición
(otra parte         es cierta.
del programa)
```

Las instrucciones de ramificación y comparación juegan un gran papel en un programa en lenguaje máquina.

El operando de las instrucciones de ramificación es la dirección de la parte del programa a la que se debe pasar cuando se cumplen las condiciones. Sin embargo, el operando es sólo un vínculo entre donde está el programa y el lugar al que debe ir. Este operando sólo tiene un byte y, por esto, el rango de las instrucciones de ramificación está limitado a 128 bytes hacia atrás o 127 hacia adelante.

> **NOTA:** Esto da un total de 255 bytes, que es -por supuesto- el máximo número de valores que puede contener un byte.

El 64MON le indica si usted ha entrado una dirección fuera de rango, negándose a "ensamblar" esta instrucción particular. El 64MON permite sin embargo escribir la dirección absoluta, convirtiendo ésta al valor apropiado para el operando. Esta es una de las ventajas de usar un ensamblador. Las instrucciones de ramificación dan una gran agilidad a un programa en lenguaje máquina.

> **NOTA:** Nos es imposible explicar todas las instrucciones de ramificación una por una. Para obtener más información consulte la bibliografía en el Apéndice F.

SUBRUTINAS

Usted puede usar en lenguaje máquina (igual que en BASIC) subrutinas. La instrucción para llamar a una subrutina es JSR (Salta a subrutina), seguida de la dirección absoluta especifica.

Hay una subrutina en el sistema operativo que imprime un carácter en la pantalla. El código CBM ASCII del carácter debe encontrarse en el acumulador antes de llamar a esta subrutina. La dirección de esta subrutina es $FFD2.

Aplicando esta información podemos realizar el siguiente programa que imprime "HI" en la pantalla:

```
.A 1400 LDA #$48       – carga el código CBM ASCII de "H"
.A 1402 JSR $FFD2      – lo imprime
.A 1405 LDA #$49       – carga el código CBM ASCII de "I"
.A 1407 JSR $FFD2      – lo imprime también
.A 140A LDA #$0D       – carga el código de retorno de carro
.A 140C JSR $FFD2      – lo imprime para pasar de línea
.A 140F BRK            – vuelve al 64MON
.G 1400                – imprime "HI" en la pantalla y vuelve
                         al 64MON
```

La rutina utilizada para imprimir el carácter forma parte de la "tabla de saltos" del KERNAL. La instrucción similar a GOTO en BASIC es en lenguaje máquina JMP, que significa saltar a la dirección absoluta especificada. El KERNAL es una larga lista de subrutinas "standarizadas" que controlan TODAS las entradas y salidas del Commodore 64. Cada entrada de la "tabla de saltos" del KERNAL se dirige hacia un subrutina del sistema operativo. Esta tabla de saltos se encuentra entre las posiciones $FF84 a $FFF5 en el sistema operativo. En la sección de este manual dedicada al KERNAL se explica detalladamente el propósito, funcionamiento y requisitos de todas las subrutinas del KERNAL. Sin embargo, estamos usando algunas rutinas del mismo en esta sección para demostrarle lo efectivo que es.

Ahora vamos a aplicar los conocimientos aprendidos en un nuevo programa que colocará las instrucciones en su contexto para que vea realmente cómo se utilizan. Este programa muestra el alfabeto usando una rutina del KERNAL. La única instrucción nueva es TXA, que significa transferir el contenido del registro X al Acumulador.

```
A 1400 LDX #$41        – X=CBM ASCII de "A"
.A 1402 TXA            – A=X
.A 1403 JSR $FFD2      – imprimir carácter
.A 1406 INX            – incrementa el contador
.A 1407 CPX #$5B       – hemos pasado de la "Z"?
.A 1409 BNE $1402      – no, continúa en $1402
.A 140B BRK            – si, vuelve al 64MON
```

Para ver cómo su Commodore 64 imprime el alfabeto, escriba el ya familiar comando:

```
.G 1400
```

Los comentarios colocados al lado de cada instrucción explican de manera clara el curso del programa y su lógica. Si desea escribir un programa en lenguaje máquina hágalo primero en un papel, y después pruebe una a una las diversas secciones del ...ino.

NOTAS UTILES PARA EL PRINCIPIANTE

Una de las mejores formas de aprender lenguaje máquina es viendo programas realizados por otras personas. Constantemente se publican en revistas especializadas. Estúdielos también aunque el artículo y programa estén destinado a otro ordenador, siempre que use el microprocesador 6510 (o el 6502). Debe asegurarse que comprende la totalidad del programa que estudie. Esto requiere perseverancia, especialmente si está estudiando una nueva técnica que no había visto antes. Este ... puede incluso enfurecerle, pero si prevalece la paciencia, usted saldrá victorioso de la prueba.

Cuando haya estudiado suf... programas en lenguaje máquina usted DEBE escribir uno propio. Puede ser una utilidad para sus programas en BASIC, un juego, o cualquier tipo de programa en lenguaje máquina.

Procure utilizar todas las utilidades disponibles, ya sea las del ordenador o bien las de un programa que le permita escribir, editar y corregir errores de los programas en C/M. Un ejemplo puede ser el KERNAL, que le permite comprobar las teclas pulsadas, imprimir texto, controlar periféricos tales como discos, impresoras, modems, etc., manejar la memoria y la pantalla... Es extremadamente p...nte y fácil de usar (Vea la sección sobre el KERNAL, pág. 223).

Ventajas de escribir sus programas en lenguaje máquina:

1. Velocidad–El lenguaje máquina es cientos, y en algunos casos miles de veces más rápido que un lenguaje de alto nivel como el BASIC.
2. Seguridad–Un programa en lenguaje máquina puede ser totalmente "impermeable", es decir, el usuario puede hacer UNICAM...E lo que el programa le permita, y nada más. Con un programa en BASIC debe prestar mucha atención para evitar que el usuario no interrumpa el programa entrando por ejemplo un cero que cause:

```
?DIVISION BY ZERO ERROR IN 830

READY
■
```

En esencia, las posibilidades del ordenador sólo se pueden maximizar programando en lenguaje máquina.

ACERCAMIENTO A UNA TAREA IMPORTANTE

Cuando intente desarrollar un programa serio debe "pensar" como un ordenador, ...ntando descubrir de forma casi inconsciente todo lo que pasará al ejecutarse ...instrucción. Debe ... el ... de ... y una vez iniciado, es una buena idea escribirlo en papel. Use diagramas de bloques para indicar el uso de la memoria, los módulos o subrutinas requeridos, el curso del programa, etc. Imagine que desea realizar un juego de ruleta en lenguaje máquina. Usted deberá plantearlo aproximadamente de la siguiente forma:

- Mostrar el título
- Preguntar si el jugador necesita instrucciones.
- SI–mostrarlas–Después empezar el juego
- NO Empezar el juego
- INICIO inicializar todo lo necesario
- Mostrar el tablero de ruleta
- Tomar las apuestas
- Hacer girar la rueda
- Detener lentamente la bola
- Comprobar las apuestas con el numero que ha salido
- Informar al jugador
- Le queda dinero al jugador?
- SI–volver al paso 6
- NO Informar al jugador y volver a INICIO

Estas son las principales líneas del programa. Cada módulo es aproximado, puede modificarlos si lo cree conveniente. Si se encuentra con un gran problema, divida el módulo en partes lo más pequeñas posible y pruébelas individualmente, luego únalo todo. Piense que no hay casi nada imposible.

Sin embargo, la única forma de adquirir soltura en estos procesos es la PRACTICA. Practique y no abandone a la primera. Sus esfuerzos se verán ampliamente recompensados.

JUEGO DE INSTRUCCIONES DEL MICROPROCESADOR MCS6510 ORDEN ALFABETICO

ADC	Suma la memoria al acumulador con acarreo
AND	"AND" de la memoria con el acumulador
ASL	Cambia el bit de la izquierda (memoria o acumulador)
BCC	Salta si está desactivado el acarreo
BCS	Salta si está activado el acarreo
BEQ	Salta si el resultado es cero
BIT	Compara los bits de la memoria con el acumulador
BMI	Salta si el resultado es negativo
BNE	Salta si el resultado no es cero
BPL	Salta si el resultado es positivo
BRK	Fuerza una interrupción
BVC	Salta si no hay desbordamiento
BVS	Salta si hay desbordamiento

CLC	Borra la bandera de acarreo
CLD	Borra el modo decimal
CLI	Borra el bit de desactiación de interrupción
CLV	Borra la bandera de desbordamiento
CMP	Compara memoria con acumulador
CPX	Compara memoria con registro X
CPY	Compara memoria con registro Y
DEC	Decrementa la memoria una unidad
DEX	Decrementa una unidad el registro X
DEY	Decrementa una unidad el registro Y
EOR	"OR exclusivo" de la memoria con el acumulador
INC	Incrementa la memoria en uno
INX	Incrementa el registro X en uno
INY	Incrementa el registro Y en uno
JMP	Salta a una nueva dirección
JSR	Salta a nueva dirección guardando dirección de retorno
LDA	Carga el acumulador con la memoria
LDX	Carga el registro X con la memoria
LDY	Carga el registro Y con la memoria
LSR	Cambia el bit de la derecha (memoria o acumulador)
NOP	No opera
ORA	"OR" de la memoria con el acumulador
PHA	Pone el acumulador en la pila
PHP	Pone el registro de estado en la pila
PLA	Saca el acumulador de la pila
PLP	Saca el registro de estado de la pila
ROL	Desplaza un bit a la izquierda (memoria o acumulador)
ROR	Desplaza un bit a la derecha (memoria o acumulador)
RTI	Vuelve de la interrupción
RTS	Vuelve de la subrutina
SBC	Resta la memoria del acumulador con acarreo en resta
SEC	Activa la bandera de acarreo
SED	Activa el modo decimal
SEI	Desactiva el estado de interrupción
STA	Guarda el acumulador en la memoria
STX	Guarda el registro X en memoria
STY	Guarda el registro Y en memoria

TAX	Transfiere el acumulador al registro X
TAY	Transfiere el acumulador al registro Y
TSX	Transfiere el puntero de pila al registro X
TXA	Transfiere el registro X al acumulador
TXS	Transfiere el registro X al puntero de pila
TYA	Transfiere el registro Y al acumulador

En este sumario se utilizarán las siguientes convenciones:

A	Acumulador
X, Y	Registros de índice
M	Memoria
P	Registro de estado del procesador
S	Puntero de pila
√	Cambio
-	No cambio
+	Suma
∧	AND lógico
−	Resta
∀	"OR EXCLUSIVO" lógico
↑	Transfiere de la pila
↓	Transfiere a la pila
→	Transfiere a
←	Transfiere desde
V	OR lógico
PC	Contador de programa
PCH	Contador de programa alto
PCL	Contador de programa bajo
OPER	Operando
#	Modo de direccionamiento inmediato

NOTA: Al final de cada tabla se encuentra entre paréntesis un número de referencia (Ref: XX) del libro "MCS6500 MICROCOMPUTER FAMILY PROGRAMMING MANUAL" en donde se define y detalla la instrucción a fondo.

ADC

ADC

Suma la memoria al acumulador con acarreo

Operación: A + M + C → A, C

(Ref.: 2.2.1)

N Z C I D V
√ √ √ – – √

Modo de Direc.	Formato en ensamblador	Código Operan.	Num. Bytes	Num. Ciclos
Inmediato	ADC →Oper.	69	2	2
Pág. Cero	ADC Oper.	65	2	3
Pág. Cero, X	ADC Oper., X	75	2	4
Absoluto	ADC Oper.	6D	3	4
Absoluto, X	ADC Oper., X	7D	3	4*
Absoluto, Y	ADC Oper., Y	79	3	4*
(Indir., X)	ADC (Oper., X)	61	2	6
(Indir.), Y	ADC (Oper.), Y	71	2	5*

* Suma 1 si se cambia de página.

AND

AND

AND lógico con el acumulador

Operación: A ∧ M → A

(Ref.: 2.2.3.0)

N Z C I D V
√ √ – – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Inmediato	AND →Oper.	29	2	2
Pág. Cero	AND Oper.	25	2	3
Pág. Cero, X	AND Oper., X	35	2	4
Absoluto	AND Oper.	2D	3	4
Absoluto, X	AND Oper., X	3D	3	4*
Absoluto, Y	AND Oper., Y	39	3	4*
(Indir., X)	AND (Oper, X)	21	2	6
(Indir.), Y	AND (Indir), Y	31	2	5

* Suma 1 si se cambia de página.

ASL

ASL

Cambia el bit izquierdo

Operación: C ← | 7 | 6 | 5 | 4 | 3 | 2 | 1 | Ø | ← Ø

(Ref.: 10.2)

N Z C I D V
√ √ √ – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Num. Ciclos
Acumulador	ASL A	0A	1	2
Pág. Cero	ASL Oper.	06	2	5
Pág. Cero, X	ASL Oper., X	16	2	6
Absoluto	ASL Oper.	0E	3	6
Absoluto, X	ASL Oper., X	1E	3	7

BCC

BCC

Salta si acarreo desactivado

Operación: Salta si C = 0

(Ref.: 4.1.1.3)

N Z C I D V
– – – – – –

Modo de Direc.	Formato en ensamblador	Codigo Operan.	Num. Bytes	Num. Ciclos
RELATIVO	BCC Oper.	90	2	2*

* Suma 1 si el salto va a la misma página.
* Suma 2 si se salta a otra página.

BCS

Salta si acarreo activado

Operación: Salta si C = 1

(Ref.: 4.1.1.4)

N Z C I D V
– – – – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
RELATIVO	BCS Oper.	B0	2	2*

* Suma 1 si se salta a la misma página.
* Suma 2 si se salta a otra página.

BEQ

Salta si el resultado es cero

Operación: Salta si Z = 1

(Ref.: 4.1.1.5)

N Z C I D V
– – – – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
RELATIVO	BEQ Oper.	F0	2	2*

* Suma 1 si se salta a la misma página.
* Suma 2 si se salta a otra página.

BIT

Compara los bits de memoria con acumulador

Operación: AAM, M7 → N, M_6 → V
Los bits 6 y 7 se transfieren al registro de estado. Si el resultado de AAM es cero entonces Z = 1, sino Z = 0.

(Ref.: 4.2.1.1)

N Z C I D V
M7* √ – – M_6

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Pág. Cero	BIT Oper.	24	2	3
Absoluto	BIT Oper.	2C	3	4

BMI

Salta si el resultado es negativo

Operación: Salta si N = 1

(Ref.: 4.1.1.1)

N Z C I D V
– – – – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Relativo	BMI Oper.	30	2	2*

* Suma 1 si se salta a la misma página.
* Suma 2 si se salta a otra página.

BNE

Salta si resultado distinto de cero

Operación: Salta si Z = 0

(Ref.: 4.1.1.6)

N Z C I D V
\- - - - - -

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm Ciclos
Relativo	BNE Oper.	D0	2	2*

* Suma 1 si se salta a la misma página.
* Suma 2 si se salta a otra página.

BPL

Salta si el resultado es positivo

Operación: Salta si N = 0

(Ref.: 4.1.1.2)

N Z C I D V
\- - - - - -

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm Ciclos
Relativo	BPL Oper.	10	2	2*

* Suma 1 si se salta a la misma página.
* Suma 2 si se salta a otra página.

BRK

Fuerza una interrupción

Operación: PC + 2 ↓ P ↓

(Ref.: 9.11)

N Z C I D V
\- - - 1 - -

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Implícito	BRK	00	1	7

* El comando BRK no puede eliminarse activando I.

BVC

Salta si no hay desbordamiento

Operación: Salta si V = 0

(Ref.: 4.1.1.8)

N Z C I D V
\- - - - - -

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Relativo	BVC Oper.	50	2	2*

* Suma 1 si se salta a la misma página.
* Suma 2 si se salta a otra página.

BVS

BVS

Salta si hay desbordamiento

Operación: Salta si V = 1

(Ref.: 4.1.1.7)

N Z C I D V
– – – – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Relativo	BVS Oper.	70	2	2*

* Suma 1 si se salta a la misma página.
* Suma 2 si se salta a otra página.

CLC

CLC

Borra la bandera de acarreo

Operación: 0 –> C

(Ref.: 3.0.2)

N Z C I D V
– – 0 – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Implícito	CLC	18	1	2

CLD

CLD

Borra el modo decimal

Operación: 0 → D

(Ref.: 3.3.2)

N Z C I D V
– – – – 0 –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Implícito	CLD	D8	1	2

CLI

CLI

Borra el bit de desactivación de interrupción

Operación: 0 → I

(Ref.: 3.2.2)

N Z C I D V
– – – 0 – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Implícito	CLI	58	1	2

CLV

CLV

Borra la bandera de desbordamiento

Operación: 0 → V

(Ref.: 3.6.1)

N Z C I D V
– – – – – 0

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Implícito	CLV	B8	1	2

CMP

Compara memoria con acumulador

Operación: A-M

(Ref.: 4.2.1)

N Z C I D V
√ √ √ – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Inmediato	CMP #Oper.	C9	2	2
Pág. Cero	CMP Oper.	C5	2	3
Pag. Cero, X	CMP Oper., X	D5	2	4
Absoluto	CMP Oper.	CD	3	4
Absoluto, X	CMP Oper., X	DD	3	4*
Absoluto, Y	CMP Oper., Y	D9	3	4*
(Indir., X)	CMP (Oper., X)	C1	2	6
(Indir.), Y	CMP (Oper.), Y	D1	2	5*

* Suma 1 si se cambia de página

CPX

Compara memoria con registro X

Operación: X-M

(Ref.: 7.8)

N Z C I D V
√ √ √ – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Inmediato	CPX #Oper.	E0	2	2
Pág. Cero	CPX Oper.	E4	2	3
Absoluto	CPX Oper.	EC	3	4

CPY

Compara memoria con registro Y

Operación: Y-M

(Ref.: 7.9)

N Z C I D V
√ √ √ – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Inmediato	CPY #Oper.	C0	2	2
Pág. Cero	CPY Oper.	C4	2	3
Absoluto	CPY Oper.	CC	3	4

DEC

Decrementa la memoria en una unidad

Operación: M-1 → M

(Ref.: 10.7)

N Z C I D V
√ √ – – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Pág. Cero	DEC Oper.	C6	2	3
Pág. Cero, X	DEC Oper., X	D6	2	6
Absoluto	DEC Oper.	CE	3	6
Absoluto, X	DEC [illegible]	DE	3	7

DEX

DEX

Decrementa el registro X en una unidad

Operación: X–1 → X

(Ref.: 7.6)

N Z C I D V
√ √ – – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Implícito	DEX	CA	1	2

DEY

DEY

Decrementa el registro y en una unidad

Operación: Y–1 ⟶ Y

(Ref.: 7.7)

N Z C I D V
√ √ – – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Implícito	DEY	88	1	2

EOR

EOR

"Or exclusivo" de la memoria con el acumulador

Operación: A ⊻ M → A

(Ref.: 2.2.3.2)

N Z C I D V
√ √ – – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Inmediato	EOR #Oper.	49	2	2
Pág. Cero	EOR Oper.	45	2	3
Pág. Cero, X	EOR Oper., X	55	2	4
Absoluto	EOR Oper.	4D	3	4
Absoluto, X	EOR Oper., X	5D	3	4*
Absoluto, Y	EOR Oper., Y	59	3	4*
(Indir., X)	EOR (Oper., X)	41	2	6
(Indir.), Y	EOR (Oper), Y	51	2	5*

* Suma 1 si se salta de página.

INC

INC

Incrementa la memoria en una unidad

Operación: M + 1 → M

(Ref.: 10.6)

N Z C I D V
√ √ – – – –

Modo de Direc.	Formato en ensamblador	Codigo Operan.	Núm. Bytes	Núm. Ciclos
Pág. Cero	INC Oper.	E6	2	5
Pág. Cero	INC Oper., X	F6	2	6
Absoluto	INC Oper.	EE	3	6
Absoluto, X	INC Oper., X	FE	3	7

INX — INX

Incrementa el registro X en una unidad

Operación: X + 1 → X

(Ref.: 7.4)

N Z C I D V
√ √ – – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Implícito	INX	E8	1	2

INY — INY

Incrementa el registro y en una unidad

Operación: Y + 1 → Y

(Ref.: 7.5)

N Z C I D V
√ √ – – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Implícito	INY	C8	1	2

JMP — JMP

Salta a una nueva dirección

Operación: (PC + 1) → PCL
(PC + 2) → PCH

(Ref.: 4.0.2 y 9.8.1)

N Z C I D V
– – – – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Absoluto	JMP Oper.	4C	3	3
Indirecto	JMP (Oper.)	6C	3	5

JSR — JSR

Salta a nueva dirección guardando dirección de retorno

Operación: PC + 2 ↓ , (PC + 1) → PCL
(PC + 2) → PCH

(Ref.: 8.1)

N Z C I D V
– – – – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Absoluto	JSR Oper.	20	3	6

LDA

Carga el acumulador con la memoria

Operación: M → A

(Ref.: 2.1.1)

N Z C I D V
√ √ – – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Inmediato	LDA #Oper.	A9	2	2
Pág. Cero	LDA Oper.	A5	2	3
Pág. Cero, X	LDA Oper., X	B5	2	4
Absoluto	LDA Oper.	AD	3	4
Absoluto, X	LDA Oper., X	BD	3	4*
Absoluto, Y	LDA Oper., Y	B9	3	4*
(Indir., X)	LDA (Oper., X)	A1	2	6
(Indir.), Y	LDA (Oper.), Y	B1	2	5*

* Suma 1 si se salta de página.

LDX

Carga el registro X con la memoria

Operación: M → X

(Ref.: 7.0)

N Z C I D V
√ √ – – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Inmediato	LDX #Oper.	A2	2	2
Pág. Cero	LDX Oper.	A6	2	3
Pág. Cero, Y	LDX Oper., Y	B6	2	4
Absoluto	LDX Oper.	AE	3	4
Absoluto, Y	LDX Oper., Y	BE	3	4*

* Suma 1 si se salta de página.

LDY

Carga el registro Y con la memoria

Operación: M → Y

(Ref.: 7.1)

N Z C I D V
√ √ – – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Inmediato	LDY #Oper.	A0	2	2
Pág. Cero	LDY Oper.	A4	2	3
Pág. Cero, X	LDY Oper., X	B4	2	4
Absoluto	LDY Oper.	AC	3	4
Absoluto, X	LDY Oper., X		3	4*

* Suma 1 si se salta de página.

LSR

Cambia el bit de la derecha (memoria o acumulador)

Operación: 0 → | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | → C

(Ref.: 10.1)

N Z C I D V
0 √ √ – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Num. Bytes	Num. Ciclos
Acumulador	LSR A	4A	1	2
Pág. Cero	LSR Oper.	46	2	5
Pág. Cero, X	LSR Oper., X	56	2	6
Absoluto	LSR Oper.	4E	3	6
Absoluto, X	LSR Oper., X	5E	3	7

NOP NOP

No se opera

Operación: No se opera (Se pierden 2 ciclos)

N Z C I D V
\- - - - - -

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Implícito	NOP	EA	1	2

ORA ORA

"Or" de memoria con acumulador

Operación: A V M → A

(Ref.: 2.2.3.1)

N Z C I D V
√ √ - - - -

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Inmediato	ORA #Oper.	09	2	2
Pág. Cero	ORA Oper.	05	2	3
Pág. Cero, X	ORA Oper., X	15	2	4
Absoluto	ORA Oper.	0D	3	4
Absoluto, X	ORA Oper., X	1D	3	4*
Absoluto, Y	ORA Oper., Y	19	3	4*
(Indir., X)	ORA (Oper., X)	01	2	6
(Indir.), Y	ORA (Oper.), Y	11	2	5

* Suma 1 si se salta de página.

PHA PHA

Pone el acumulador en la pila

Operación: A ↓

(Ref.: 8.5)

N Z C I D V
\- - - - - -

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Implícito	PHA	48	1	3

PHP PHP

Pone el registro de estado en la pila

Operación: P ↓

(Ref.: 8.11)

N Z C I D V
\- - - - - -

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Implícito	PHP	08	1	3

PLA PLA

Coge el acumulador de la pila

Operación: A ↑

(Ref.: 8.6)

N Z C I D V
√ √ - - - -

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Implícito	PLA	68	1	4

PLP

*Coge el registro de **estado de la pila***

Operación: P ↑

(Ref.: 8.12)

N Z C I D V
De la pila

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Implícito	PLP	28	1	4

ROL

Desplaza un bit a la izquierda (memoria o acumulador)

Operación: [7|6|5|4|3|2|1|0] ← C

(Ref.: 10.3)

N Z C I D V
√ √ √ – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Acumulador	ROL A	2A	1	2
Pág. Cero	ROL Oper.	26	2	5
Pág. Cero, X	ROL Oper., X	36	2	6
Absoluto	ROL Oper.	2E	3	6
Absoluto, X	ROL Oper., X	3E	3	7

ROR

Desplaza un bit a la derecha (memoria o acumulador)

Operación: → [C] → [7|6|5|4|3|2|1|0] →

(Ref.: 10.4)

N Z C I D V
√ √ √ – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Acumulador	ROR A	6A	1	2
Pág. Cero	ROR Oper.	66	2	5
Pág. Cero, X	ROR Oper., X	76	2	6
Absoluto	ROR Oper.	6E	3	6
Absoluto, X	ROR Oper., X	7E	3	7

NOTA: La instrucción ROR está disponible en el microprocesador desde junio de 1976.

RTI

Vuelve de una interrupción

Operación: P ↑ PC ↑

(Ref.: 9.6)

N Z C I D V
De la pila

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Implícito	RTI	40	1	6

RTS

RTS

Vuelve de una subrutina

Operación: PC ↑, PC + 1 → PC

(Ref.: 8.2)

N Z C I D V
– – – – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Num. Bytes	Num. Ciclos
Implícito	RTS	60	1	6

SBC

SBC

Resta la memoria del acumulador con acarreo

Operación: A–M–$\overline{C}$ → A

(Ref.: 2.2.2)

N Z C I D V
√ √ √ – – √

Modo de Direc.	Formato en ensamblador	Código Operan.	Num. Bytes	Num. Ciclos
Inmediato	SBC #Oper	E9	2	2
Pág. Cero	SBC Oper.	E5	2	3
Pág. Cero, X	SBC Oper., X	F5	2	4
Absoluto	SBC Oper.	ED	3	4
Absoluto, X	SBC Oper., X	FD	3	4*
Absoluto, Y	SBC Oper., Y	F9	3	4*
(Indir., X)	SBC (Oper., X)	E1	2	6
(Indir.), Y	SBC (Oper.), Y	F1	2	5*

* Suma 1 si se salta de página.

SEC

SEC

Activa la bandera de acarreo

Operación: 1 → C

(Ref.: 3.0.1)

N Z C I D V
– – 1 – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Num. Ciclos
Implícito	SEC	38	1	2

SED

SED

Activa el modo decimal

Operación: 1 –> D

(Ref.: 3.3.1)

N Z C I D V
– – – – 1 –

Modo de Direc.	Formato en ensamblador	Código Operan.	Num. Bytes	Num. Ciclos
Implícito	SED	F8	1	2

SEI

SEI

Activa el estado de interrupción inhibida

Operación: 1 → I

(Ref.: 3.2.1)

N Z C I D V
– – – 1 – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Num. Bytes	Num. Ciclos
Implícito	SEI	78	1	2

STA STA

Almacena el acumulador en memoria

Operación: $A \rightarrow M$

(Ref.: 2.1.2)

N Z C I D V
— — — — — —

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Pág. Cero	STA Oper.	85	2	3
Pág. Cero, X	STA Oper., X	95	2	4
Absoluto	STA Oper.	8D	3	4
Absoluto, X	STA Oper., X	9D	3	5
Absoluto, Y	STA Oper., Y	99	3	5
(Indir. X)	STA (Oper., X)	81	2	6
(Indir.), Y	STA (Oper.), Y	91	2	6

STX STX

Almacena el registro X en memoria

Operación: $X \rightarrow M$

(Ref.: 7.2)

N Z C I D V
— — — — — —

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Pág. Cero	STX Oper.	86	2	3
Pág. Cero, Y	STX Oper., Y	96	2	4
Absoluto	STX Oper.	8E	3	4

STY STY

Almacena el contenido del registro y en memoria

Operación: $Y \rightarrow M$

(Ref.: 7.3)

N Z C I D V
— — — — — —

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Pág. Cero	STY Oper.	84	2	3
Pág. Cero, X	STY Oper., X	94	2	4
Absoluto	STY Oper.	8C	3	4

TAX TAX

Transfiere el acumulador al registro X

Operación: $A \rightarrow X$

(Ref. 7.11)

N Z C I D V
√ √ — — — —

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Implícito	TAX	AA	1	2

TAY

Transfiere el acumulador al registro Y

Operación: A → Y

(Ref.: 7.13)

N Z C I D V
√ √ – – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Implícito	TAY	A8	1	2

TSX

Transfiere el puntero de pila al registro X

Operación: S → X

(Ref. 8.9)

N Z C I D V
√ √ – – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Núm. Ciclos
Implícito	TSX	BA	1	2

TXA

Transfiere el registro X al acumulador

Operacin: X → A

(Ref.: 7.12)

N Z C I D V
√ √ – – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Num. Ciclos
Implícito	TXA	8A	1	2

TXS

Transfiere el registro X al puntero de pila

Operación: X → S

(Ref.: 8.8)

N Z C I D V
– – – – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Num. Ciclos
Implícito	TXS	9A	1	2

TYA

Transfiere el registro y al acumulador

Operación: Y → A

(Ref.: 7.14)

N Z C I D V
√ √ – – – –

Modo de Direc.	Formato en ensamblador	Código Operan.	Núm. Bytes	Num. Ciclos
Implícito	TYA	98	1	2

MODO DE DIRECCIONAMIENTO DE LAS INSTRUCCIONES Y RELACION DE LOS TIEMPOS DE EJECUCION
(En ciclos de reloj)

	Acumulador	Inmediato	Página Cero	Página Cero, X	Página Cero, Y	Absoluto	Absoluto, X	Absoluto, Y	Implícito	Relativo	(Indirecto, X)	(Indirecto), Y	Absoluto Ind.
ADC	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
AND	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
ASL	2	.	5	6	.	6	7	.	.	.	.	.	.
BCC	.	.	.	.	.	.	.	.	.	2**	.	.	.
BCS	.	.	.	.	.	.	.	.	.	2**	.	.	.
BEQ	.	.	.	.	.	.	.	.	.	2**	.	.	.
BIT	.	.	3	.	.	4	.	.	.	.	.	.	.
BMI	.	.	.	.	.	.	.	.	.	2**	.	.	.
BNE	.	.	.	.	.	.	.	.	.	2**	.	.	.
BPL	.	.	.	.	.	.	.	.	.	2**	.	.	.
BRK	.	.	.	.	.	.	.	.	.	.	.	.	.
BVC	.	.	.	.	.	.	.	.	.	2**	.	.	.
BVS	.	.	.	.	.	.	.	.	.	2**	.	.	.
CLC	.	.	.	.	.	.	.	.	2	.	.	.	.
CLD	.	.	.	.	.	.	.	.	2	.	.	.	.
CLI	.	.	.	.	.	.	.	.	2	.	.	.	.
CLV	.	.	.	.	.	.	.	.	2	.	.	.	.
CMP	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
CPX	.	2	3	.	.	4	.	.	.	.	.	.	.
CPY	.	2	3	.	.	4	.	.	.	.	.	.	.
DEC	.	.	5	6	.	6	7	.	.	.	.	.	.
DEX	.	.	.	.	.	.	.	.	2	.	.	.	.
DEY	.	.	.	.	.	.	.	.	2	.	.	.	.
EOR	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
INC	.	.	5	6	.	6	7	.	.	.	.	.	.
INX	.	.	.	.	.	.	.	.	2	.	.	.	.
INY	.	.	.	.	.	.	.	.	2	.	.	.	.
JMP	.	.	.	.	.	3	.	.	.	.	.	.	5

* Suma un ciclo si se indexa saltando de página.
** Suma un ciclo si se bifurca. Suma uno adicional si se salta de página.

MODO DE DIRECCIONAMIENTO DE LAS INSTRUCCIONES Y RELACION DE LOS TIEMPOS DE EJECUCION
(En ciclos de reloj)

	Acumulador	Inmediato	Página Cero	Página Cero, X	Página Cero, Y	Absoluto	Absoluto, X	Absoluto, Y	Implícito	Relativo	(Indirecto, X)	(Indirecto), Y	Absoluto Ind.
JSR	.	.	.	.	.	6	.	.	.	.	.	.	.
LDA	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
LDX	.	2	3	.	4	4	.	4*	.	.	.	.	.
LDY	.	2	3	4	.	4	4*	.	.	.	.	.	.
LSR	2	.	5	6	.	6	7	.	.	.	.	.	.
NOP	.	.	.	.	.	.	.	.	2	.	.	.	.
ORA	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
PHA	.	.	.	.	.	.	.	.	3	.	.	.	.
PHP	.	.	.	.	.	.	.	.	3	.	.	.	.
PLA	.	.	.	.	.	.	.	.	4	.	.	.	.
PLP	.	.	.	.	.	.	.	.	4	.	.	.	.
ROL	2	.	5	6	.	6	7	.	.	.	.	.	.
ROR	2	.	5	6	.	6	7	.	.	.	.	.	.
RTI	.	.	.	.	.	.	.	.	6	.	.	.	.
RTS	.	.	.	.	.	.	.	.	6	.	.	.	.
SBC	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
SEC	.	.	.	.	.	.	.	.	2	.	.	.	.
SED	.	.	.	.	.	.	.	.	2	.	.	.	.
SEI	.	.	.	.	.	.	.	.	2	.	.	.	.
STA	.	.	3	4	.	4	5	5	.	.	6	6	.
STX	.	.	3	.	4	4	.	.	.	.	.	.	.
STY	.	.	3	4	.	4	.	.	.	.	.	.	.
TAX	.	.	.	.	.	.	.	.	2	.	.	.	.
TAY	.	.	.	.	.	.	.	.	2	.	.	.	.
TSX	.	.	.	.	.	.	.	.	2	.	.	.	.
TXA	.	.	.	.	.	.	.	.	2	.	.	.	.
TXS	.	.	.	.	.	.	.	.	2	.	.	.	.
TYA	.	.	.	.	.	.	.	.	2	.	.	.	.

* Suma un ciclo si se indexa saltando de página.
** Suma un ciclo si se bifurca. Suma uno adicional si se salta de página.

00 – BRK
01 – ORA– (Indirecto, X)
02 – *
03 – *
04 – *
05 – ORA– Pág. Cero
06 – ASL– Pág. Cero
07 – *
08 – PHP
09 – ORA– Inmediato
0A – ASL– Acumulador
0B – *
0C – *
0D – ORA–Absoluto
0E – ASL– Absoluto
0F – *
10 – BPL
11 – ORA– (Indirecto), Y
12 *
13 – *
14 – *
15 – ORA– Pág. Cero, X
16 – ASL–Pág. Cero, X
17 – *
18 – CLC
19 – ORA– Absoluto, Y
1A – *
1B – *
1C – *
1D – ORA– Absoluto, X
1E – ASL– Absoluto, X
1F – *
20 – JSR
21 – AND– (Indirecto, X)
22 – *
23 – *
24 – BIT– Pág. Cero
25 – AND– Pág. Cero
26 – ROL– Pág. Cero
27 – *
28 – PLP
29 – AND– Inmediato
2A – ROL– Acumulador
2B – *
2C – BIT– Absoluto
2D – AND– Absoluto
2E – ROL– Absoluto
2F – *
30 – BMI
31 – AND (Indirecto), Y
32 – *
33 – *
34 – *
35 – AND– Pág. Cero, X
36 – ROL– Pág. Cero, X
37 – *
38 – SEC
39 – AND– Absoluto, Y
3A – *
3B – *
3C – *
3D – AND– Absoluto, X
3E – ROL– Absoluto, X
3F – *
40 – RTI
41 – EOR– (Indirecto, X)
42 – *
43 – *
44 – *
45 – EOR– Pág. Cero
46 – LSR– Pág. Cero
47 – *
48 – PHA
49 – EOR– Inmediato
4A – LSR– Acumulador
4B – *
4C – JMP– Absoluto
4D – EOR– Absoluto
4E – LSR– Absoluto
4F – *
50 – BVC
51 – EOR– (Indirecto), Y
52 – *
53 – *
54 – *
55 – EOR– Pág. Cero
56 – LSR– Pág. Cero
57 – *
58 – CLI
59 – EOR– Absoluto, Y
5A – *
5B – *
5C – *
5D – EOR– Absoluto, X

* Estos códigos se reservan para expansiones futuras. En el presente no están asignados a ninguna instrucción.

5E – LSR– Absoluto, X
5F – *
60 – RTS
61 – ADC– (Indirecto, X)
62 – *
63 – *
64 – *
65 – ADC– Pág. Cero
66 – ROR– Pág. Cero
67 – *
68 PLA
69 – ADC– Inmediato
6A – ROR– Acumulador
6B – *
6C – JMP– Indirecto
6D – ADC– Absoluto
6E – ROR– Absoluto
6F – *
70 BVS
71 – ADC– (Indirecto), Y
72 – *
73 – *
74 – *
75 – ADC– Pág. Cero
76 – ROR– Pág. Cero
77 – *
78 – SEI
79 – ADC– Absoluto, Y
7A – *
7B – *
7C – *
7D – ADC– Absoluto, X
7E – ROR– Absoluto, X
7F – *
80 – *
81 – STA– (Indirecto, X)
82 – *
83 – *
84 – STY– Pág. Cero
85 – STA– Pág. Cero
86 – STX– Pág. Cero
87 – *
88 – DEY
89 – *
8A – TXA
8B – *
8C – STY– Absoluto
8D – STA · Absoluto
8E – STX– Absoluto
8F – *
90 – BCC
91 – STA– (Indirecto), Y
92 – *
93 – *
94 – STY– Pág. Cero, X
95 – STA– Pág. Cero, X
96 – STX– Pag. Cero, Y
97 – *
98 – TYA
99 – STA– Absoluto, Y
9A – TXS
9B – *
9C – *
9D – STA– Absoluto, X
9E – *
9F – *
A0 – LDY– Inmediato
A1 – LDA– (Indirecto, X)
A2 – LDX– Inmediato
A3 – *
A4 – LDY– Pág. Cero
A5 – LDA– Pág. Cero
A6 – LDX– Pág. Cero
A7 – *
A8 – TAY
A9 – LDA– Inmediato
AA – TAX
AB – *
AC – LDY– Absoluto
AD – LDA– Absoluto
AE – LDX– Absoluto
AF – *
B0 – BCS
B1 – LDA– (Indirecto), Y
B2 – *
B3 – *
B4 – LDY– Pág. Cero, X
B5 – LDA– Pág. Cero, X
B6 – LDX– Pág. Cero, Y
B7 – *
B8 – CLV
B9 – LDA– Absoluto, Y
BA – TSX
BB – *

* Estos códigos se reservan para expansiones futuras. En el presente no están asignados a ninguna instrucción.

BC – LDY– Absoluto, X
BD – LDA– Absoluto, X
BE – LDX– Absoluto, Y
BF – *
C0 – CPY– Inmediato
C1 – CMP– (Inmediato, X)
C2 – *
C3 – *
C4 – CPY– Pág. Cero
C5 – CMP– Pág. Cero
C6 – DEC– Pág. Cero
C7 – *
C8 – INY
C9 – CMP– Inmediato
CA – DEX
CB – *
CC – CPY– Absoluto
CD – CMP– Absoluto
CE – DEC– Absoluto
CF – *
D0 – BNE
D1 – CMP– (Indirecto), Y
D2 – *
D3 – *
D4 – *
D5 – CMP– Pág. Cero, X
D6 – DEC– Pág. Cero, X
D7 – *
D8 – CLD
D9 – CMP– Absoluto, Y
DA – *
DB – *
DC – *
DD – CMP– Absoluto, X
DE – CMP– Absoluto, X
DF – *
E0 – CPX– Inmediato
E1 – SBC– (Indirecto, X)
E2 – *
E3 – *
E4 – CPX– Pág. Cero
E5 – SBC– Pág. Cero
E6 – INC– Pág. Cero
E7 – *
E8 – INX
E9 – SBC– Inmediato
EA – NOP
EB – *
EC – CPX– Absoluto
ED – SBC– Absoluto
EE – INC– Absoluto
EF – *
F0 – BEQ
F1 – SBC– (Inmediato), Y
F2 – *
F3 – *
F4 – *
F5 – SBC– Pág. Cero, X
F6 – INC– Pág. Cero, X
F7 – *
F8 – SED
F9 – SBC– Absoluto, Y
FA – *
FB – *
FC – *
FD – SBC– Absoluto, X
FE – INC– Absoluto, X
FF – *

* Estos códigos se reservan para expansiones futuras. En el presente no están asignados a ninguna instrucción.

ORGANIZACION DE LA MEMORIA DEL COMMODORE 64

El Commodore 64 posee 64K bytes de RAM. Dispone también de 20K bytes de [illegible] que contienen el sistema operativo, el intérprete BASIC y el juego de caracteres [illegible]ard. También tiene 4K de entradas/salidas para periféricos. ¿Cómo es [illegible] acceso a toda esta memoria si un ordenador con un bus de direcciones de [illegible] normalmente solo puede direccionar 64K.?
[illegible] se encuentra en el propio microprocesador 6510.

En el chip se encuentra un port de E/S. Este port se usa para controlar que RAM, ROM o E/S debe aparecer en determinadas posiciones de la memoria del sistema. Este port se usa también para controlar el Datassette, por lo que es importante que únicamente se vean afectados los bits correctos.
El port de E/S del 6510 se encuentra en la dirección 1. El registro de dirección de datos de dicho port se encuentra en la posición 0. El port se controla igual que cualquier otro port del sistema... el registro de dirección de datos indica si el port es de entrada o de salida, y la transferencia de datos ocurre en el mismo port.
A continuación se definen las líneas del port de control del 6510:

NOMBRE	BIT	DIRECCION	DESCRIPCION
LORAM	0	SALIDA	Control para RAM/ROM en $A000–SBFFF (BASIC)
HIRAM	1	SALIDA	Control para RAM/ROM en $E000–$FFFF (KERNAL)
CHAREN	2	SALIDA	Control de la ROM E/S en $D000–$DFFF
	3	SALIDA	Línea de escritura del cassette
	4	ENTRADA	Interruptor del sentido del cassette
	5	SALIDA	Control del motor del cassette

El valor adecuado para el registro de dirección de datos es:

BITS 5 4 3 2 1
1 0 1 1 1

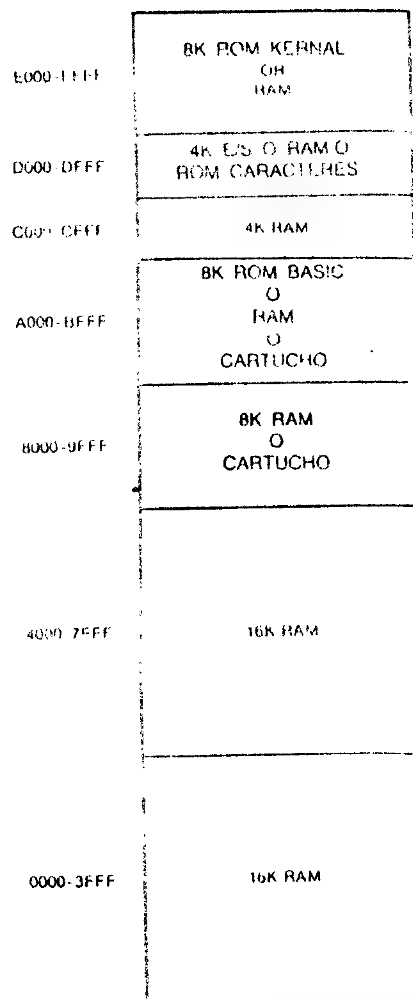
(Donde 1 es salida y 0 entrada)

Esto da un valor de 47 en decimal. El Commodore 64 ajusta automáticamente este valor en el registro de datos.
Las líneas de control realizan, en general, las funciones indicadas en su descripción. Sin embargo, algunas combinaciones de líneas de control se usan ocasionalmente para lograr una configuración particular de memoria.
LORAM (bit 0) Esta línea controla normalmente los 8K del intérprete BASIC en ROM. Esta línea está ALTA para la operación con BASIC. Si se coloca BAJA, la ROM del BASIC desaparece de la memoria y es reemplazada por 8K de RAM en las posiciones $A000–$BFFF.
HIRAM (bit 1) Esta línea controla normalmente los 8K del KERNAL en ROM. Cuando esta línea está ALTA (normal) el KERNAL está presente. Si se programa BAJA el KERNAL desaparecerá de la memoria para dejar paso a 8K de RAM desde $E000 a $FFFF.
CHAREN (bit 2) Se usa para controlar la ROM de 4K bytes que contiene el juego de caracteres. Desde el punto de vista del procesador, esta ROM ocupa el mismo espacio que la zona de E/S ($D000–$DFFF). Cuando esta línea está a 1 (normal), en el espacio de direccionamiento del procesador aparecen los registros de E/S, no siendo accesible el juego de caracteres en ROM. Cuando esta línea está a 0 es el

juego de caracteres el que aparece en lugar de las E/S, que no son accesibles. El microprocesador solo necesita acceder al juego de caracteres cuando se desean trasladar los mismos de ROM a RAM. Es necesario tener un especial cuidado para realizar esta operación. (Vea la sección de CARACTERES PROGRAMABLES en el capítulo de GRAFICOS). CHAREN no tiene efecto en una configuración de memoria que no precise E/S. En su lugar aparecerá RAM desde $D000 a $DFFF.

En cualquier mapa de memoria conteniendo ROM, la escritura (POKE) en una posición de memoria correspondiente a ROM se efectuará en la RAM existente debajo de la ROM. Por supuesto, la lectura (PEEK) de una posición correspondiente a ROM dará como resultado el valor contenido en ROM, no en RAM.

MAPA DE MEMORIA BASICO DEL COMMODORE 64



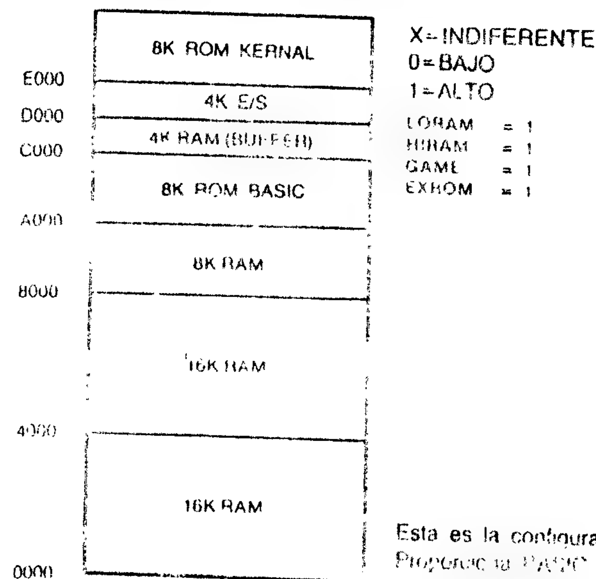
ESQUEMA DE LAS ENTRADAS/SALIDAS

D000-D3FF	VIC (Controlador de video)	1K Bytes
D400-D7FF	SID (Sintetizador de sonido)	1K Bytes
D800-DBFF	RAM de color	1K Nybbles
DC00-DCFF	CIA 1 (Teclado)	256 Bytes
DD00-DDFF	CIA 2 (Bus serie/RS-232/Port usuario)	256 Bytes
DE00-DEFF	E/S abiertas #1 (Activación CP/M)	256 Bytes
DF00-DFFF	E/S abiertas #2 (Disco)	256 Bytes

Las dos E/S abiertas son para usos generales de E/S, cartuchos de E/S (como el IEEE), activación del cartucho Z-80 (CP/M en opción) y para enlazar con unidades de disco de gran rapidez y bajo coste.
Está previsto el inicio automático de los programas contenidos en los cartuchos de expansión del C-64. El programa en cartucho se ejecuta si los 9 primeros bytes del mismo (a partir de la posición 32768 ($8000)) contienen datos específicos. Los dos primeros bytes deben contener la dirección de inicio de la ejecución del programa en cartucho. Los próximos 2 bytes (32770-32771) ($8002-$8003) deben contener el vector de inicio usado por el programa en cartucho. Los próximos 3 byes deben ser las letras CBM con el bit 7 a 1 en cada letra. Los últimos dos bytes deben contener los dígitos "80" en PET ASCII.

MAPAS DE MEMORIA DEL COMMODORE 64

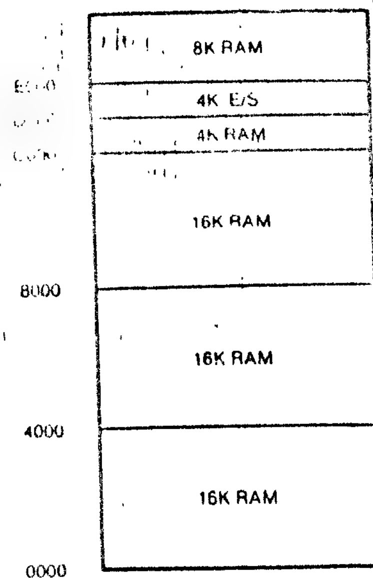
Las siguientes tablas le muestran las distintas configuraciones de memoria disponibles en el Commodore 64, el estado de las líneas que seleccionan cada configuración y el probable uso de la misma.



X=INDIFERENTE
0=BAJO
1=ALTO

LORAM = 1
HIRAM = 1
GAME = 1
EXROM = 1

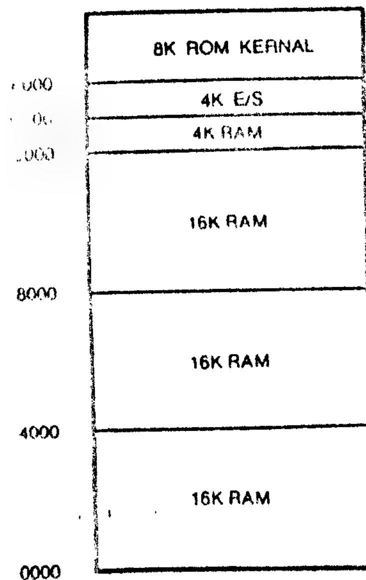
Esta es la configuración normal.
Proporciona la BASIC



X=INDIFERENTE
0=BAJO
1=ALTO

LORAM = 1
HIRAM = 0
GAME = 1
EXROM = X
O
LORAM = 1
HIRAM = 0
GAME = 0
(Los caracteres en ROM no están disponibles en esta configuración)
EXROM = 0

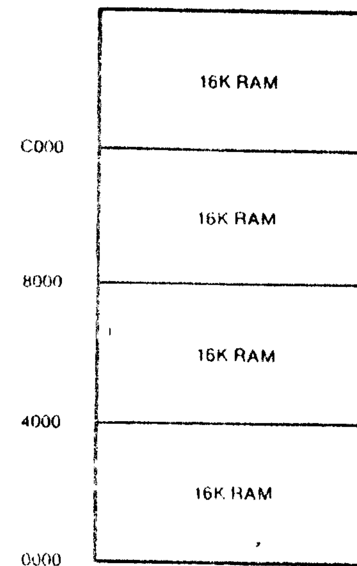
Memoria disponible: 60K de RAM y 4K E/S. Las rutinas para gestionar las E/S deben ser escritas por el usuario.



X=INDIFERENTE
0=BAJO
1=ALTO

LORAM = 0
HIRAM = 1
GAME = 1
EXROM = X

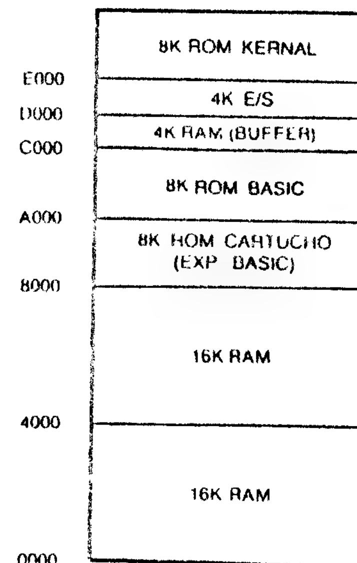
(Este mapa se puede usar con otros lenguajes (incluyendo CP/M), con un total de 52K RAM usuario, E/S y rutinas para manejar las E/S)



X=INDIFERENTE
0=BAJO
1=ALTO

LORAM = 0
HIRAM = 0
GAME = 1
EXROM = X
O
LORAM = 0
HIRAM = 0
GAME = X
EXROM = 0

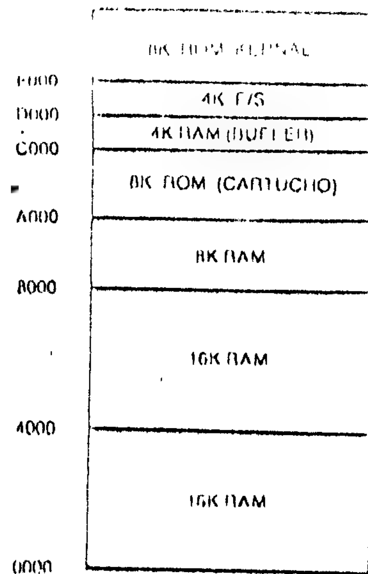
(Este mapa le da acceso a los 64K de RAM. En cualquier operación de E/S será necesario permitir el acceso del procesador a la zona E/S)



X=INDIFERENTE
0=BAJO
1=ALTO

LORAM = 1
HIRAM = 1
GAME = 0
EXROM = 0

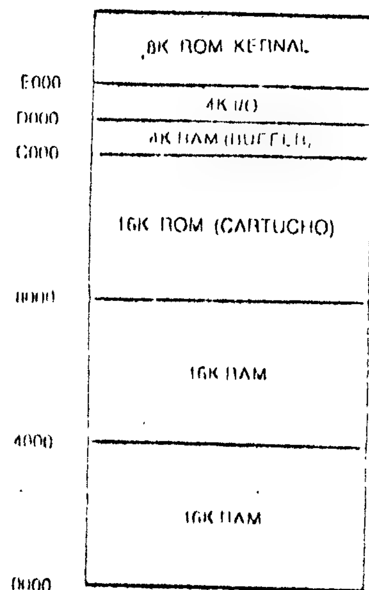
(Esta es la configuración standard de un sistema en BASIC con expansión en ROM del BASIC 32K RAM usuario y ha[illegible] 8K ROM de extensión del BASIC)



X=INDIFERENTE
0=BAJO
1=ALTO

LORAM	=	0
HIRAM	=	1
GAME	=	0
EXROM	=	0

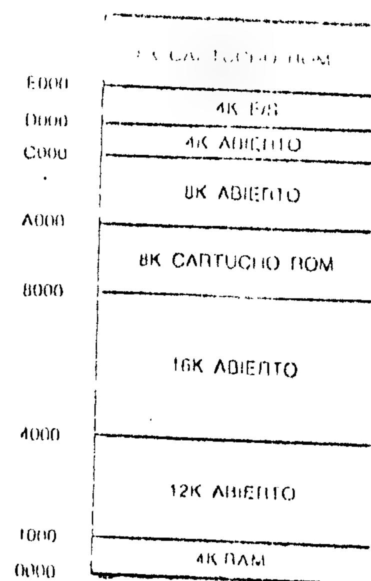
(Este mapa le proporciona 40K contiguos de RAM usuario y hasta 8K de ROM conectable para aplicaciones basadas en ROM que no requieran el BASIC)



X=INDIFERENTE
0=BAJO
1=ALTO

LORAM	=	1
HIRAM	=	1
GAME	=	0
EXROM	=	0

(Este mapa le proporciona 32K contiguos de RAM usuario y hasta 16K de ROM en cartucho para aplicaciones que no requieran el BASIC. (Proceso de texto, otros lenguajes, etc.))



0=BAJO
1=ALTO

LORAM	=	X
HIRAM	=	X
GAME	=	0
EXROM	=	1

(Este es el mapa de memoria para video juegos ULTIMAX. Advierta que los 2K de expansión RAM, si se requieren, son necesarios fuera del C-64 y cualquier RAM presente en el cartucho es ignorada)

EL KERNAL

Uno de los principales problemas de cara a los programadores en el campo de los microordenadores es el miedo a las modificaciones que el fabricante pueda hacer en el ordenador. Los programas en lenguaje máquina no durarían mucho tiempo, obligando a continuas revisiones. Commodore ha desarrollado un método para proteger a los autores de software llamado KERNAL.

Básicamente, el KERNAL es una tabla de bifurcaciones standarizada de rutinas de entradas, salidas y manejo de la memoria del sistema operativo. Las direcciones de cada rutina varían con cada actualización del sistema, pero la tabla de saltos del KERNAL también se modifica. Si los programas en lenguaje máquina utilizan las rutinas del sistema sólo a través del KERNAL será muy fácil modificar los programas para adaptarlos a cualquier ordenador Commodore. El KERNAL es el sistema operativo de su Commodore 64. Todas las entradas, salidas y manejos de memoria están controladas por el KERNAL.

Para simplificar los programas en lenguaje máquina, y para evitar que queden obsoletos por futuras modificaciones del sistema operativo del comodore 64, el KERNAL dispone de una tabla de saltos para su utilización. Aprovechando al máximo las 39 rutinas de E/S y utilidades disponibles desde la tabla, no sólo ahorrará tiempo, sino que se facilitará enormemente la conversión de programas entre ordenadores Commodore.

La tabla de saltos se encuentra en la última página de la memoria del sistema, en memoria de sólo lectura (ROM).

Para utilizar esta tabla KERNAL, primero se han de preparar los parámetros que ne-

cesita la rutina para trabajar. Después se hace un JSR al lugar apropiado dè la tabla. Después de realizar la función específica, el KERNAL transfiere nuevamente el con-trol al programa en lenguaje máquina del usuario. Dependiendo de la rutina que se utilice, algunos registros pueden pasar parámetros a su programa. Los registros particulares de las rutinas KERNAL se detallan en la descripción individual de las subrutinas.

Una buena pregunta sería: ¿Por qué se ha de utilizar siempre la tabla de direcciona-mientos? ¿Por qué no se puede hacer que la instrucción JSR salte directamente a la rutina deseada? La tabla tiene su razón de ser en que si el KERNAL o el BASIC se modifican, los programas seguirán funcionando. En versiones futuras del sistema podrán cambiar las direcciones de cada rutina, pero la tabla seguirá funcionando correctamente.

ACTIVIDADES DEL KERNAL DURANTE LA PUESTA EN MARCHA

1) Al conectarse el ordenador, el KERNAL ajusta primero el puntero de pila (stack), y borra el modo decimal.
2) El KERNAL comprueba entonces la presencia de un cartucho ROM con ejecu-ción automática en la posición $8000 (32768). Si está presente, se suspende la inicialización normal y se transfiere el control al código del cartucho. Si no se en-cuentra ROM de ejecución automática el proceso de inicialización continúa.
3) Después, el KERNAL inicializa las E/S. El bus serie es inicializado. Ambas CIA 6526 son ajustadas a los valores correctos para el funcionamiento normal del te-clado, y se activa el reloj de 60 Hz. El chip de sonido (SID) es limpiado. Se selec-ciona el mapa de memoria BASIC standard y se cierra el motor del cassette.
4) Ahora el KERNAL ejecuta una comprobación de RAM, ajustando los punteros de principio y fin de memoria. También se inicializa la página cero y el buffer del cassette.
La rutina de comprobación de RAM no es destructiva y se inicia en la posición $0300, continuando hacia arriba. El puntero de fin de RAM se ajusta cuando se encuentra la primera posición que no corresponde a RAM. El límite inferior de memoria se ajusta siempre a $0800, y la pantalla se sitúa siempre a partir de $0400.
5) Por último, el KERNAL realiza estas otras actividades. Se colocan los valores normales en los vectores de E/S. Se forma la tabla indirecta de saltos en la pági-na 3. Se borra la pantalla y todas las variables del editor de pantalla son iniciali-zadas. Entonces se usa el indirecto en $A000 para inicializar el BASIC.

COMO UTILIZAR EL KERNAL

Cuando se escriben programas en lenguaje máquina es conveniente usar las ruti-nas que ya forman parte del sistema operativo para operaciones de E/S, acceso al reloj interno, manejo de memoria y operaciones similares. Es un esfuerzo innecesa-rio escribir estas rutinas de nuevo estando disponible en su Commodore 64. Un fácil acceso al sistema operativo ayuda a programar más rápidamente en lenguaje máquina.

Como ya se ha dicho, el KERNAL es una tabla de saltos. Es exactamente una rela-ción de instrucciones JMP a rutinas del sistema operativo.

Para utilizar una rutina del KERNAL se han de realizar todos los preparativos que la misma requiera... Si la rutina precisa que se realice antes otra rutina KERNAL, se ha de hacer. Si la rutina precisa que un determinado número se encuentre en el acumulador, el numero debe estar en él. De lo contrario hay pocas posibilidades de que el programa funcione correctamente.

Una vez se hayan hecho todos los preparativos, se debe llamar a la rutina por medio de la instrucción JSR. Todas las rutinas KERNAL a las que se tiene acceso estan estructuradas como subrutinas, finalizando con la instrucción RTS. Cuando la rutina KERNAL finaliza su tarea se devuelve el control al programa en la instrucción si-guiente a la JSR.

Alguna de las rutinas del KERNAL, devuelven códigos de error en el registro de es-tado o en el acumulador en caso de problemas. Es buena norma comprobarlo. Si se ignora un error devuelto, el resto del programa puede quedar totalmente inservible. Estos son los pasos a dar para utilizar el KERNAL:

1) Preparación
2) Llamada a la rutina
3) Comprobacion de errores

En la descripción de las rutinas del KERNAL se utilizaran las siguientes convencio-nes:

- **NOMBRE DE FUNCION:** Nombre de la rutina del KERNAL.
- **DIRECCION DE LLAMADA:** Es la dirección de llamada a la rutina KERNAL (en hexadecimal).
- **REGISTROS DE COMUNICACION:** Los registros que aparezcan en este epigra-fe son los que se usan para pasar parámetros desde y hacia las rutinas del KERNAL.
- **RUTINAS DE PREPARACION:** Algunas rutinas del KERNAL requieren una pre-paración de datos antes de usarse. Las rutinas necesarias apareceran en este apar-tado.
- **DEVOLUCION DE ERRORES:** El retorno de una rutina del KERNAL con el aca-rreo activado indica que se ha producido un error. El acumulador contiene el núme-ro de este error.
- **NECESIDADES DE PILA:** Es el número de bytes de la pila (el stack) que serán utilizados por la rutina del KERNAL.
- **DESCRIPCION:** Es una descripción general de la rutina, incluyendo ejemplos de funcionamiento.

A continuación se relacionan todas las rutinas del KERNAL con sus nombres, direc-ciones de llamada en decimal y hexadecimal y función de cada una de ellas.

RUTINAS DEL KERNAL UTILIZABLES POR EL USUARIO

NOMBRE	DIRECCION		FUNCION
	HEX	DECIMAL	
ACPTR	$FFA5	65445	Acepta un byte del port serie.
CHKIN	$FFC6	65478	Abre un canal de entrada.
CHKOUT	$FFC9	65481	Abre un canal de salida.
CHRIN	$FFCF	65487	Toma un carácter de un canal.
CHROUT	$FFD2	65490	Envía un carácter a un canal.
CIOUT	$FFA8	65448	Envía un byte al port serie.
CINT	$FF81	65409	Inicializa el editor de pantalla.
CLALL	$FFE7	65511	Cierra todos los canales y ficheros.
CLOSE	$FFC3	65475	Cierra un fichero lógico especificado.
CLRCHN	$FFCC	65484	Cierra los canales de entrada y salida.
GETIN	$FFE4	65512	Coge un carácter del buffer de teclado.
IOBASE	$FFF3	65523	Devuelve la dirección de base de los dispositivos de E/S.
IOINIT	$FF84	65412	Inicializa las E/S.
LISTEN	$FFB1	65457	Maneja dispositivos del Bus Serie.
LOAD	$FFD5	65493	Carga a RAM desde un periférico.
MEMBOT	$FF9C	65436	Lee/Ajusta el inicio de la memoria.
MEMTOP	$FF99	65493	Lee/Ajusta el final de la memoria.
OPEN	$FFCO	65472	Abre un fichero lógico.
PLOT	$FFF0	65520	Lee/Ajusta la posición X, Y del cursor.
RAMTAS	$FF87	65415	Inicializa RAM, coloca buffer del cassette, coloca la pantalla a partir de $0400.
RDTIM	$FFDE	65502	Lee el reloj de tiempo real.
READST	$FFB7	65463	Lee la palabra de Estado de E/S.
RESTOR	$FF8A	65418	Restituye los valores normales de E/S.
SAVE	$FFD8	65496	Guarda RAM en un periférico.
SCNKEY	$FF9F	65439	Rastrea el teclado.
SCREEN	$FFED	65517	Devuelve la organización X,Y de la pantalla.
SECOND	$FF93	65427	Envía la dirección secundaria después de LISTEN.
SETLFS	$FFBA	65466	Ajusta la dirección lógica y la dirección secundaria.
SETMSG	$FF90	65424	Controla los mensajes del KERNAL.
SETNAM	$FFBD	65469	Coloca el nombre de fichero.
SETTIM	$FFDB	65499	Ajusta el reloj de tiempo real.
SETTMO	$FFA2	65442	Ajusta el tiempo del Bus serie
STOP	$FFE1	65505	Rastrea la tecla de STOP.
TALK	$FFB4	65460	Ordena al periférico que envíe datos.
TKSA	$FF96	65430	Envía dirección secundaria después de TALK.
UDTIM	$FFEA	65514	Incrementa el reloj de tiempo real.
UNLSN	$FFAE	65454	Envía al bus serie la orden de terminar la "escucha".
UNTLK	$FFAB	65451	Envía al bus serie la orden de terminar el envío de datos.
VECTOR	$FF8D	65421	Lee/Ajusta los vectores de E/S.

B-1. Nombre de la función: ACPTR

Propósito: Toma datos del Bus Serie
Dirección de llamada: $FFA5 (65445)
Registros de comunicación: A
Rutinas preparatorias: TALK, TKSA
Errores devueltos: Ver READST
Necesidades de pila: 13
Registros afectados: A, X

Descripción: Esta rutina se usa para tomar información de un periférico a través del Bus Serie. (Por ejemplo el disco). Toma un byte de datos del Bus colocándolo en el acumulador. Se debe haber llamado previamente a la rutina TALK para que el periférico envíe datos al Bus. Si el periférico precisa un comando adicional se le debe enviar a través de la rutina TKSA antes de llamar a ésta. Los errores se devuelven en la palabra de estado.

Cómo usarla:

0) Enviar un comando al periférico para que se prepare para enviar datos al Bus Serie. (Utilizar las rutinas TALK y TKSA).
1) Llamar a esta rutina. (Mediante JSR).
2) Procesar los datos.

EJEMPLO:

Toma un byte del Bus:

```
JSR ACPTR
STA DATA
```

B-2. Nombre de función: CHKIN

Propósito: Abrir un canal de entrada
Dirección de llamada: $FFC6 (65478)
Registros de comunicación: (OPEN)
Rutinas preparatorias: 3, 5, 6
Necesidades de pila: Ninguna
Registros afectados: A, X

Descripción: Un fichero lógico abierto previamente con la rutina OPEN del KERNAL se puede definir como canal de entrada mediante esta rutina. Naturalmente, el periférico de dicho canal debe ser de entrada. Si no fuese así se produciría un error, abortando la rutina.
Si se utiliza un dispositivo de entrada que no sea el teclado se debe llamar a esta rutina antes de poder utilizar las rutinas de entrada de datos CHRIN o GETIN. Si se desea entrar datos a través del teclado y no hay abiertos otros canales de entrada, no se necesita llamar a esta rutina ni a la OPEN.
Cuando se utiliza esta rutina con un periférico del Bus Serie se envía directamente la dirección de comunicación (y la dirección secundaria si se ha especificado en la rutina OPEN).

Cómo usarla:

0) Abrir el fichero lógico si es necesario. (Vea la descripción).
1) Cargar el registro X con el número de fichero lógico utilizado.
2) Llamar a esta rutina (utilizando el comando JSR).

Errores posibles:

#3: Fichero no abierto
#5: Periférico no presente
#6: El fichero no es de entrada.

EJEMPLO:

Preparación para entrada desde el fichero lógico 2.

```
LDX #2
JSR CHKIN
```

B-3. Nombre de función: CHKOUT

Propósito: Abrir un canal de salida
Dirección de llamada: $FFC9 (65481)
Registros de comunicación: X
Rutinas preparatorias: (OPEN)
Errores devueltos: 3, 5, 7
Necesidades de pila: Ninguna
Registros afectados: A, X

Descripción: Permite definir como canal de salida un fichero abierto mediante la rutina OPEN. Por lo tanto, el periférico correspondiente debe ser de salida, de lo contrario se produciría un error y la rutina sería abortada.
Se debe llamar a esta rutina antes de enviar datos al periférico de salida a no ser que éste sea la pantalla. Si se desea usar la pantalla y no hay definidos otros canales de salida es innecesario llamar a esta rutina y abrir el canal mediante la OPEN.
Cuando se usa esta rutina para abrir un fichero de salida se envía automáticamente la dirección de escucha especificada en la rutina OPEN, y la dirección secundaria si hiciera falta.

Cómo usarla:

RECUERDE: Esta rutina no es necesaria si se desea utilizar la pantalla como periférico.

0) Use la rutina del KERNAL OPEN para especificar el número de fichero lógico, la dirección de escucha y –si es preciso la dirección secundaria.
1) Cargue en el registro X el número de fichero lógico utilizado en la rutina OPEN.
2) Llame a la rutina mediante JSR.

EJEMPLO:

```
LDX #3          ;DEFINE EL FICHERO LOGICO 3 COMO CANAL DE SALIDA
JSR CHKOUT
```

Errores posibles:

#3: Fichero no abierto
#5: Periférico no presente
#7: El fichero no es de salida

B-4. Nombre de función: CHRIN

Propósito: Toma un carácter de un canal de entrada
Dirección de llamada: $FFCF (65487)
Registros de comunicación: A
Rutinas preparatorias: (OPEN, CHKIN)
Errores devueltos: Ver READST
Necesidades de pila: Ninguna
Registros afectados: A, X

Descripción: Esta rutina toma un byte de datos de un canal preparado como entrada mediante la rutina CHKIN. Si no se ha definido otro canal de entrada, los datos se toman del teclado. El byte de datos se envía al acumulador. El canal permanece abierto después de la llamada.
La entrada desde el teclado se trata de una forma especial. Primero se activa el cursor y se hace parpadear hasta que se pulsa la tecla RETURN en el teclado. Todos los caracteres de la línea (hasta 88) se guardan en el buffer de entrada del BASIC. Entonces se pueden tomar uno a uno mediante esta rutina. Al encontrar el RETURN se ha procesado toda la línea. La siguiente vez que se llama a esta rutina se repite el proceso, parpardeando el cursor.

Cómo usarla:

DESDE EL TECLADO:

1) Tomar un byte de datos llamando a esta rutina mediante JSR
2) Almacenar el byte de datos.
3) Comprobar si es el último byte (si es un retorno de carro).
4) Si no lo es, volver al paso 1.

EJEMPLO:

```
      LDY #00       ;PREPARA EL REGISTRO Y PARA GUARDAR
                    LOS DATOS
RD    JSR CHRIN     ;GUARDA EL Y BYTE EN LA Y POSICION DEL AREA
                    ;DESTINADA A ALMACENAR LOS DATOS
      INY
      CMP #CR       ;¿ES UN RETORNO DE CARRO?
      BNE RD        ;NO, TOMAR OTRO BYTE DE DATOS
```

EJEMPLO:

```
JSR CHRIN
STA DATA
```

DESDE OTROS PERIFERICOS:

0) Utilizar las rutinas OPEN y CHKIN del KERNAL.
1) Llamar a esta rutina.
2) Guardar los datos.

EJEMPLO:

```
JSR CHRIN
STA DATA
```

Nombre de la función: CHROUT

Propósito: Envía un carácter
Dirección de llamada: $FFD2 (65490)
Registros de comunicación: A
Rutinas de preparación: (CHKOUT, OPEN)
Errores devueltos: Ver READST
Necesidades de pila: 8+
Registros afectados: A

Descripción: Esta rutina envía un carácter a un canal de salida ya preparado. Hay que preparar previamente el canal de salida mediante las rutinas OPEN y CHKOUT. Si no se hiciese, los datos se enviarán a la pantalla. El byte de datos a enviar se carga en el acumulador, llamando después a esta rutina, enviándose al periférico indicado. [illegible] anal sigue abierto después de llamar a esta rutina.

NOTA: Hay que tener mucho cuidado al utilizar esta rutina para enviar datos a periféricos conectados al Bus Serie puesto que los datos se enviarán a todos los periféricos de salida abiertos en él. Para [illegible] se deberá cerrar todos los canales que no se deban utilizar [illegible]

Cómo usarla:

0) Utilizar la rutina KERNAL CHKOUT si es necesaria (Vea descripción).
1) Cargar el dato a enviar en el acumulador.
2) Llamar a esta rutina.

EJEMPLO:

```
;DUPLICA LA INSTRUCCION BASIC CMD4, "A";
LDX #4                  ;FICHERO LOGICO 4
JSR                     ;ABRE EL CANAL DE SALIDA
LDA #'A
JSR CHROUT              ;ENVIA UN CARACTER
```

B-6. Nombre de la función: CIOUT

Propósito: Transmite un byte por el Bus Serie
Dirección de llamada: $FFA8 (65448)
Registros de comunicación: A
Rutinas preparatorias: LISTEN [SECOND]
Errores devueltos: Ver READST
Necesidades de pila: 5
Registros afectados: Ninguno

Descripción: Esta rutina se utiliza para enviar información a periféricos conectados al Bus Serie. Una llamada a esta rutina pondrá un byte de datos en el Bus Serie. Antes de llamar a esta rutina se debe ejecutar la rutina LISTEN para preparar al periférico para recibir datos del Bus Serie. (Si el periférico necesita dirección secundaria, se le debe enviar mediante la rutina SECOND). El acumulador se carga con el byte que se quiere enviar. El periférico debe haber sido preparado mediante LISTEN o se producirá un error. La rutina tiene un buffer de un carácter. (Se guarda el byte anterior al que se quiere enviar). Cuando finaliza la transmisión de datos llamando a la rutina UNLSN, el carácter del buffer se envía junto con un EOI (Fin o identificación). Entonces se envía el comando UNLSN al periférico.

Cómo usarla:

0) Llame primero a la rutina del KERNAL LISTEN (y si es necesario también a la rutina SECOND).
1) Cargue el acumulador con un byte de datos.
2) Llame a esta rutina para enviar el byte de datos.

EJEMPLO:

```
LDA #'X                 ;ENVIA UNA X AL BUS SERIE
JSR CIOUT
```

B-7. Nombre de función: CINT

Propósito: Inicializar al editor de pantalla y el chip de vídeo 6567.
Dirección de llamada: $FF81 (65409)
Registros de comunicación: Ninguno
Rutinas preparatorias: Ninguna
Errores devueltos: Ninguno
Necesidades de pila: 4
Registros afectados: A, X, Y

Descripción: Esta rutina ajusta el chip de vídeo 6567 del Commodore 64 para la operación normal. También se inicializa el editor de pantalla del KERNAL. Esta rutina debe ser utilizada por los programas en cartucho.

Cómo usarla:

1) Llamar a esta rutina.

EJEMPLO:

```
JSR CINT
JMP RUN                    ;EMPEZAR EJECUCION
```

B-8. Nombre de la funcion: CLALL

Propósito: Cierra todos los ficheros.
Dirección de llamada: $FFE7 (65511)
Registros de comunicación: Ninguno
Rutinas preparatorias: Ninguna
Errores devueltos: Ninguno
Necesidades de pila: 11
Registros afectados: A, X

Descripción: Esta rutina cierra todos los ficheros abiertos. Al llamarla se inicializan los punteros de la tabla de ficheros, cerrando todos los ficheros. Ademas, inicializa los canales de E/S.

Cómo usarla:

1) Llamar a esta rutina.

EJEMPLO:

```
JSR CLALL      ;CIERRA TODOS LOS FICHEROS Y COLOCA LOS
               VALORES
               ;NORMALES EN LAS E/S.
JMP RUN        ;EMPIEZA LA EJECUCION
```

B-9. Nombre de función: CLOSE

Propósito: Cierra un fichero lógico.
Dirección de llamada: $FFC3 (65475)
Registros de comunicación: A
Rutinas preparatorias: Ninguna
Errores devueltos: 0, 240 (Ver READST)
Necesidades de pila: 2+
Registros afectados: A, X, Y

Descripción: Esta rutina se utiliza para cerrar un fichero lógico después de realizar las operaciones de E/S deseadas. Se llama después de cargar en el acumulador el número de fichero lógico que se desea cerrar. (El mismo número utilizado al abrirlo con la rutina OPEN).

Cómo usarla:

1) Cargar el número de fichero lógico a cerrar en el acumulador.
2) Llamar a esta rutina.

EJEMPLO:

```
;CIERRA EL CANAL 15
LDA #15
JSR CLOSE
```

B-10. Nombre de función: CLRCHN

Propósito: Limpiar los canales de E/S
Dirección de llamada: $FFCC (65484)
Registros de comunicación: Ninguno
Rutinas preparatorias: Ninguna
Errores devueltos: Ninguno
Necesidades de pila: 9
Registros afectados: A, X

Descripción: Se utiliza para limpiar todos los canales abiertos y que vuelvan a sus valores originales. Se utiliza normalmente después de abrir canales de E/S (como el disco o la cinta) y haberlos utilizado en operaciones de E/S. El periferico de salida normal es el 3 (pantalla), y el de entrada el 0 (teclado).
Si uno de los canales a cerrar es el port serie, se envia primero la señal UNTALK para limpiar el canal de entrada o la UNLISTEN si el canal es de salida. Si no se utiliza esta rutina (dejando activa la escucha del Bus Serie) algunos periféricos pueden recibir simultáneamente los datos del COMMODORE 64. Uno de los medios de sacar partido de esto es dejar al disco con TALK y la impresora con LISTEN. De esta forma se pueden obtener directamente impresiones de ficheros en disco. Esta rutina se ejecuta automaticamente cuando se llama a la rutina CLALL.

Cómo usarla:

1) Llame a esta rutina mediante JSR.

EJEMPLO:

JSR CLRCHN

B-11. Nombre de la función: GETIN

Propósito: Tomar un carácter.
Dirección de llamada: $FFE4 (65058)
Registros de comunicación: A
Rutinas preparatorias: CHKIN, OPEN
Errores devueltos: Vea READST
Necesidades de pila: 7+
Registros afectados: A (X, Y)

Descripción: Si el canal es el teclado, esta rutina coge un carácter de la cola del teclado colocando su valor en ASCII en el acumulador. Si la cola está vacía, el valor cargado en el acumulador será 0. Los caracteres se ponen automáticamente en la cola gracias a una de las interrupciones, que se encarga de rastrear el teclado y llamar a la rutina SCNKEY. El buffer del teclado puede almacenar hasta 10 caracteres. Si se pulsan más teclas, se perderan si el buffer está lleno. Si el canal es el RS-232, entonces se usa el registro A para colocar el carácter leído. Vea READST para comprobar la validación. Si el canal es el Bus serie, el cassette o la pantalla, llame a la rutina BASIN.

Cómo usarla:

1) Llame esta rutina con JSR.
2) Compruebe si el acumulador contiene 0. (Buffer vacío).
3) Procese los datos.

EJEMPLO:

```
;ESPERA UN CARACTER
WAIT JSR GETIN
CMP #0
BEQ WAIT
```

B-12. Nombre de la función: IOBASE

Propósito: Definir una página de memoria de E/S
Dirección de llamada: $FFF3 (65523)
Registros de comunicación: X, Y
Rutinas preparatorias: Ninguna
Errores devueltos:
Necesidades de pila: 2
Registros afectados: X, Y

Descripción: Esta rutina coloca en los registros X e Y la dirección donde se encuentra la memoria mapeada de E/S. Esta dirección puede utilizarse para acceder a la memoria mapeada de E/S de los periféricos del Commodore 64. Es el número de las direcciones de comienzo de la página en la que se encuentra el registro de E/S deseado. El registro X contiene el byte bajo de la dirección, mientras que el registro Y contiene el byte alto.
Esta rutina sirve para permitir la compatibilidad entre el Commodore 64 y el VIC-20, así como con los futuros modelos de Commodore. Si las direcciones de E/S en un programa escrito en lenguaje máquina se toman mediante esta rutina, será compatible con futuras versiones del Commodore 64, del KERNAL o del BASIC.

Cómo usarla:

1) Llamar a esta rutina mediante JSR.
2) Guardar los registros X e Y en direcciones consecutivas.
3) Cargar el registro Y con la dirección.
4) Acceder a la dirección de E/S.

EJEMPLO:

```
;COLOCA EL REGISTRO DE DIRECCION DE DATOS DEL PORT DE USUARIO
A 0 (ENTRADA)
JSR IOBASE
STX POINT        ;AJUSTA LOS REGISTROS DE BASE
STY POINT+1
LDY #2
LDA #0           ;VALOR DEL DDR DEL PORT USUARIO
STA (POINT), Y   ;COLOCA EL DDR A 0
```

B-13. Nombre de función: IOINIT

Propósito: Inicializar los periféricos de E/S
Dirección de llamada: $FF84 (65412)
Registros de comunicación: Ninguno
Rutinas preparatorias: Ninguna
Errores devueltos:
Requerimientos de pila: Ninguno
Registros afectados: A, X, Y

Descripción: Esta rutina inicializa todos los periféricos de E/S. Normalmente se llama como parte de la inicialización de un programa en cartucho.

EJEMPLO:

JSR IOINIT

B-14. Nombre de la función: LISTEN

Propósito: Prepara un periférico para recibir datos
Dirección de llamada: $FFB1 (65457)
Registros de comunicación: A
Rutinas preparatorias: Ninguna
Errores devueltos: Ver READST
Necesidades de pila: Ninguna
Registros afectados: A

Descripción: Esta rutina prepara un periférico en el Bus Serie para recibir datos. Se debe cargar el acumulador con un número de periférico entre 0 y 31 antes de llamar a esta rutina. LISTEN ejecuta un OR bit a bit para convertirlo en la dirección de escucha, y después transmite el dato como comando al bus serie. El periférico especificado pasa a modo de "escucha", y queda listo para recibir información.

Cómo usarla:

1) Cargue el acumulador con el número de periférico.
2) Llame esta rutina mediante JSR

EJEMPLO:

```
;PREPARA LA RECEPCION DE DATOS DEL PERIFERICO 8
LDA #8
JSR LISTEN
```

B-15. Nombre de función: LOAD

Propósito: Cargar RAM desde un periférico
Dirección de llamada: $FFD5 (65493)
Registros de comunicación: A,X,Y
Rutinas preparatorias: SETLFS, SETNAM
Errores devueltos: 0,4,5,8,9 (Vea también ST)
Necesidades de pila: Ninguna
Registros afectados: A,X,Y

Descripción: Esta rutina carga en la memoria del Commodore 64 datos directamente desde un periférico de entrada. Se puede utilizar también para la verificación, comparando los datos del periférico con los residentes en memoria, sin modificar el contenido de ésta. El acumulador debe estar a 0 para operaciones de carga y a 1 para operaciones de verificación. Si el periférico de entrada se ha abierto con dirección secundaria (SA) de 0, la información de cabecera del periférico se ignorará. En este caso, los registros X e Y deben contener la dirección de inicio para la carga. Si se ha enviado al periférico la dirección secundaria 1 o 2, los datos se cargarán en memoria empezando en la dirección especificada en la cabecera. Esta rutina devuelve la posición más alta de RAM que se ha alcanzado. Antes de llamar a esta rutina se deben ejecutar las rutinas SETLFS y SETNAM.

Cómo usarla:

0) Llamar a las rutinas SETLFS y SETNAM. Si se desea una carga relocatable, la rutina SETLFS debe enviar una dirección secundaria de 0.
1) Coloque en el acumulador un 0 para cargar o un 1 para verificar.
2) Si se desea una dirección de carga, colóquela en los registros X e Y.
3) Llame a la rutina mediante JSR.

EJEMPLO:

```
;CARGA UN FICHERO DESDE CINTA
LDA  #DEVICE           ;PONE EL NUMERO DE PERIFERICO
LDX  #FILENO           ;PONE EL NUMERO DE FICHERO LOGICO
LDY  #CMD1             ;COLOCA LA DIRECCION SECUNDARIA 1
JSR  SETLFS
LDA  #NAME1-NAME       ;CARGA EN EL ACUMULADOR EL NUMERO DE
                        CARACTERES DEL NOMBRE DEL FICHERO
LDX  #<NAME            ;CARGA EN X E Y LA DIRECCION DONDE SE INICIA
                        EL NOMBRE DEL FICHERO
LDY  #>NAME
JSR  SETNAM
LDA  #0                ;INDICA CARGA (0= CARGA/1=VERIFICACION)
LDX  #$FF              ;INICIO DE LA CARGA DONDE INDIQUE LA
                        CABECERA DE LA CINTA
LDY  #$FF
JSR  LOAD
STX  VARTAB            ;FIN DE CARGA
STY  VARTAB+1
NAME                   ;BYT "FILE NAME"
NAME1                  ;
```

B-16. Nombre de función: MEMBOT

Propósito: Ajustar el inicio de memoria
Dirección de llamada: $FFC9 (65436)
Registros de comunicación: X,Y
Rutinas preparatorias: Ninguna
Errores devueltos: Ninguno
Necesidades de pila: Ninguna
Registros afectados: X,Y

Descripción: Esta rutina se usa para ajustar el inicio de la memoria. Si el bit de acarreo del acumulador está a 1 cuando se llama a esta rutina, se devuelve en los registros X e Y el puntero del byte más bajo de la memoria. En el Commodore 64 sin expansión ROM el valor inicial de este puntero es $0800 (2048 decimal). Si el bit de acarreo del acumulador está a cero, los valores de X e Y se transfieren a los bytes alto y bajo respectivamente del puntero de inicio de la memoria RAM.

Cómo se usa:

Para leer el inicio de memoria
1) Colocar el bit de acarreo a 1
2) Llamar a esta rutina*

Para ajustar el inicio de la memoria
1) Colar el bit de acarreo a cero
2) Llamar a esta rutina*

* Antes de llamar a estas rutinas para modificar los límites de memoria se deben colocar en X e Y el valor del nuevo puntero que se desea (N. del T.)

EJEMPLO:

```
;MOVER UNA PAGINA HACIA ARRIBA EL INICIO DE LA MEMORIA
SEC                ;LEE EL INICIO DE LA MEMORIA
JSR MEMBOT
INY
CLC                ;COLOCA EL NUEVO VALOR COMO INICIO DE
                   LA MEMORIA
JSR MEMBOT
```

B-17. Nombre de la función: MEMTOP

Propósito: Ajustar el final de la memoria
Dirección de llamada: $FF99 (65433)
Registros de comunicación: X e Y
Rutinas preparatorias: Ninguna
Errores devueltos: Ninguno
Necesidades de pila: 2
Registros afectados: X,Y

Descripción: Esta rutina se utiliza para ajustar el final de la memoria RAM. Al llamar a esta rutina con el bit de acarreo a 1, se cargará el puntero de fin de memoria en los registros X e Y. Cuando se llama con el bit de acarreo a 0, el contenido de los registros X e Y se cargará en el puntero de fin de memoria, modificándolo.

EJEMPLO:

```
;LIBERA EL BUFFER DEL RS-232
SEC
JSR MEMTOP         ;LEE EL FINAL DE LA MEMORIA
DEX
CLC
JSR MEMTOP         ;AJUSTA EL LIMITE DE LA MEMORIA.
```

B-18. Nombre de la función: OPEN

Propósito: Abre un fichero lógico
Dirección de llamada: $FFCO (65472)
Registros de comunicación: Ninguno
Rutinas preparatorias: SETLFS, SETNAM
Errores devueltos: 1, 2, 4, 5, 6, 240, READST
Necesidades de pila: Ninguna
Registros afectados: A, X, Y

Descripción: Esta rutina se usa para abrir un fichero lógico. Una vez activado el fichero lógico, puede ser utilizado para operaciones de E/S. Muchas de las rutinas del KERNAL de E/S llaman a esta rutina para crear el fichero lógico con el que van a

trabajar. Esta rutina no requiere argumentos, pero deben haberse ejecutado previamente las rutinas SETLFS y SETNAM.

Cómo se utiliza:

0) Utilizar la rutina SETLFS.
1) Utilizar la rutina SETNAM.
2) Llamar a esta rutina.

EJEMPLO:

Esta es una simulación de la sentencia BASIC: OPEN 15,8,15, "I:0"

```
LDA #NAME2-NAME    ;LONGITUD DEL NOMBRE DE FICHERO
                   PARA SETLFS
LDY #>NAME
LDX #<NAME
JSR SETNAM
LDA #15
LDX #8
LDY #15
JSR SETLFS
JSR OPEN
NAME BYT "I:0"
NAME2
```

B-19. Nombre de la función: PLOT

Propósito: Ajusta la posición del cursor
Dirección de llamada: $FFF0
Registros de comunicación: A, X, Y
Rutinas preparatorias: Ninguna
Errores devueltos: Ninguno
Necesidades de pila: 2
Registros afectados: A, X, Y

Descripción: Una llamada a esta rutina con la bandera de acarreo activada carga la posición actual del cursor en coordenadas X, Y en los registros X e Y. X es el número de columna (0–79) e Y el número de fila (0–24). Una llamada con la bandera de acarreo a cero mueve el cursor a la posición X, Y determinada por los registros X e Y.

Cómo usarla:

Lectura de la posición del cursor.
1) Activar la bandera de acarreo.
2) Llamar a esta rutina.
3) Tomar la posición X, Y de los registros X, Y respectivamente.

Modificación de la posición del cursor.
1) Desactivar la bandera de acarreo.
2) Colocar en los registros X e Y la posición deseada.
3) Llamar a esta rutina.

EJEMPLO:

```
;MUEVE EL CURSOR A LA FILA 10, COLUMNA 5 (5, 10)
LDX #10
LDY #5
CLC
JSR PLOT
```

B-20. Nombre de la función: RAMTAS

Propósito: Realizar una comprobación de RAM
Dirección de llamada: $FF87 (65415)
Registros de comunicación: A, X, Y
Rutinas preparatorias: Ninguna
Errores devueltos: Ninguno
Necesidades de pila: 2
Registros afectados: A, X, Y

Descripción: Esta rutina se usa para realizar una comprobación de RAM y ajustar el inicio y fin de la memoria RAM. También reajusta las posiciones de memoria de $0000 a $0101 y de $0200 a $03FF, ajustando el buffer del cassette y colocando la memoria de pantalla a partir de la posición $0400. Normalmente se llama a esta rutina como parte del proceso de inicialización de un programa en cartucho.

EJEMPLO:

```
JSR RAMTAS
```

B-21. Nombre de la función: RDTIM

Propósito: Leer el reloj del sistema
Dirección de llamada: $FFDE (65502)
Registros de comunicación: A, X, Y
Rutinas preparatorias: Ninguna
Errores devueltos: Ninguno
Necesidades de pila: 2
Registros afectados: A, X, Y

Descripción: Esta rutina se usa para leer el reloj del sistema. La resolución del mismo es de 1/60 de segundo. La rutina devuelve 3 bytes. El acumulador contiene el byte más significativo, el registro X contiene el byte intermedio y el registro Y el byte menos significativo.

EJEMPLO:

```
JSR RDTIM
STY TIME
STX TIME+1
STA TIME+2
...
TIME *=*+3
```


B-22. Nombre de la función: READST

Propósito: Lee la palabra de estado.
Dirección de llamada: $FFB7 (65463)
Registros de comunicación: A
Rutinas preparatorias: Ninguna
Errores devueltos: Ninguno
Necesidades de pila: 2
Registros afectados: A

Descripción: Esta rutina devuelve el estado actual de los periféricos de E/S en el acumulador. Se utiliza normalmente después de una operación de E/S. Da información sobre el estado o errores ocurridos durante una operación de E/S.
Los bits devueltos en el acumulador contienen la siguiente información: (Vea la tabla)

POS BIT ST	VALOR BIT ST	LECTURA CASSETTE	L/E SERIE	CASSETTE VERIFY Y LOAD
0	1		E. Fuera de tiempo	
1	2		L. Fuera de tiempo	
2	4	Bloque corto		Bloque corto
3	8	Bloque largo		Bloque largo
4	16	Error de Lec. irrecuperable		Algún error
5	32	Error de comprobación		Error de comprobación
6	64	Fin de fichero	Línea EOI	
7	−128	Fin de cinta	Periférico no presente	Fin de cinta

Cómo usarla:

1) Llamar a esta rutina.
2) Decodificar la información del Acumulador

EJEMPLO:

```
;COMPROBACION DE FIN DE FICHERO DURANTE LA LECTURA
JSR READST
AND #64          ;COMPRUEBA EL BIT DE EOF (EOF=FIN DE FICHERO)
BNE EOF          ;BIFURCA EN CASO DE EOF
```

B-23. Nombre de la función: RESTOR

Propósito: Inicializa el sistema y los vectores de interrupción
Dirección de llamada: $FF8A (65418)
Rutinas preparatorias: Ninguna
Errores devueltos: Ninguno
Necesidades de pila: 2
Registros afectados: A, X, Y

Descripción: Esta rutina devuelve los valores normales a todos los vectores del sistema utilizados en rutinas KERNAL y BASIC (vea el mapa de memoria para conocer las posiciones de los vectores). La rutina VECTOR del KERNAL se utiliza para leer y modificar vectores individuales.

Cómo usarla:

1) Llamar a esta rutina.

EJEMPLO:

```
JSR RESTOR
```

Nombre de la función: SAVE

Propósito: Guardar contenido de la memoria en periféricos
Dirección de llamada: $FFD8 (65496)
Registros de comunicación: A, X, Y
Rutinas preparatorias: SETLFS, SETNAM
Errores devueltos: 5, 8, 9, READST
Necesidades de pila: Ninguna
Registros afectados: A, X, Y

Descripción: Esta rutina se encarga de guardar parte del contenido de la memoria del Commodore 64 en un periférico como el Datassette o la unidad de disco. La memoria se guarda desde una dirección indirecta de página cero especificada en el acumulador hasta la dirección especificada en los registros X e Y. Se debe utilizar las rutinas SETLFS y SETNAM antes de llamar a esta. Sin embargo, no es necesario especificar nombre de fichero si se utiliza el periférico 1 (cinta). Se producirá un error si intenta utilizar otro periférico sin asignar nombre de fichero.

> NOTA: El periférico 0 (teclado) y el 3 (pantalla) no se pueden utilizar para guardar algo en ellos. Si se intenta se producirá un error deteniéndose la rutina.

Cómo se utiliza:

0) Utilizar las rutinas SETLFS y SETNAM (a no ser que se quiera grabar en cinta sin nombre de fichero).
1) Cargar dos direcciones consecutivas de página cero con el puntero de inicio de la zona a grabar.
2) Cargar el acumulador con el byte de página cero inicio del puntero.
3) Cargar en los registros X e Y la última dirección a copiar.
4) Llamar a esta rutina.

EJEMPLO:

```
LDA #1            ;DISPOSITIVO=1:CASSETTE
JSR SETLFS
LDA #0            ;NO SE DA NOMBRE DE FICHERO
JSR SETNAM
LDA PROG          ;CARGA INICIO DE LA MEMORIA A GRABAR
STA TXTTAB        ;BYTE BAJO
LDA PROG+1
STA TXTTAB+1      ;BYTE ALTO
LDX VARTAB        ;CARGA X CON EL BYTE BAJO DE FIN DE MEMORIA
LDY VARTAB+1      ;CARGA Y CON EL BYTE ALTO DE FIN DE MEMORIA
LDA #<TXTTAB      ;CARGA EL ACUMULADO CON LA DIRECCION DE PAGINA
                  ;CERO QUE CONTIENE EL BYTE BAJO DEL PUNTERO
JSR SAVE
```

B-25. Nombre de función: SCNKEY

Propósito: Rastrear el teclado
Dirección de llamada: $FF9F (65439)
Registros de comunicación: Ninguno
Rutinas preparatorias: IOINIT
Errores devueltos: Ninguno
Necesidades de pila: 5
Registros afectados: A, X, Y

Descripción: Esta rutina rastrea el teclado comprobando si se ha pulsado alguna tecla. Es la misma rutina llamada por la rutina de interrupción. Si se ha pulsado alguna tecla, se coloca su valor ASCII en el buffer de teclado. Esta rutina es llamada sólo si se pasa a través del IRQ normal.

Cómo usarla:

1) Llame a esta rutina.

EJEMPLO:

```
GET  JSR SCNKEY     ;RASTREA EL TECLADO
     JSR GETIN      ;TOMA UN CARACTER
     CMP #0         ;ES NULO?
     BEQ GET        ;SI... RASTREA OTRA VEZ
     JSR CHROUT     ;IMPRIMALO
```

B-26. **Nombre de la función: SCREEN**

Propósito: Devuelve el formato de la pantalla
Dirección de llamada: $FFED (65517)
Registros de comunicación: X, Y
Rutinas preparatorias: Ninguna
Necesidades de pila: 2
Registros afectados: X, Y

Descripción: Esta rutina devuelve el formato de la pantalla, por ejemplo 40 columnas en X y 25 líneas en Y. Ha sido implementada en el Commodore 64 para permitir la compatibilidad de sus programas.

Cómo usarla:

1) Llamar a esta rutina.

EJEMPLO:

```
JSR SCREEN
STX MAXCOL
STY MAXROW
```

B-27. **Nombre de la función: SECOND**

Propósito: Enviar una dirección secundaria para LISTEN
Dirección de llamada: $FF93 (65427)
Registros de comunicación: A
Rutinas preparatorias: LISTEN
Errores devueltos: Vea READST
Necesidades de pila: 8
Registros afectados: A

Descripción: Esta rutina se utiliza para enviar una dirección secundaria a un periférico de E/S después de hacer una llamada a la rutina LISTEN. No puede utilizarse después de la rutina TALK.
La dirección secundaria se utiliza normalmente para informar al periférico sobre algun modo especial antes de realizar las operaciones de E/S.

Cómo usarla:

1) Cargue el acumulador con la dirección secundaria a enviar.
2) Llame a esta rutina.

EJEMPLO:

```
DIRECCION DE PERIFERICO 8 CON LA DIRECCION SECUNDARIA 15
LDA #8
JSR LISTEN
LDA #15
JSR SECOND
```


B-28. **Nombre de la función: SETLFS**

Propósito: Ajustar un fichero lógico
Dirección de llamada: $FFBA (65466)
Registros de comunicación: A,X,Y
Rutinas preparatorias: Ninguna
Errores devueltos: Ninguno
Necesidades de pila: 2
Registros afectados: Ninguno

Descripción: Esta rutina ajusta el número de fichero lógico, dirección del periférico y la dirección secundaria para otras rutinas del KERNAL.
El número de fichero lógico se utiliza por el sistema como una clave para la tabla de ficheros creada con la rutina OPEN. La dirección del periférico puede estar entre 0 y 31. El Commodore 64 usa los siguientes códigos para los dispositivos CBM:

DIRECCION	PERIFERICO
0	TECLADO
1	DATASSETTE
2	RS-232
3	PANTALLA CRT
4	IMPRESORA SERIE
8	DISCO CBM SERIE

Los números iguales o mayores a 4 se refieren al Bus Serie. Una orden a un periférico se envía como una dirección secundaria sobre el Bus serie. Si no se desea dirección secundaria, el registro Y debe estar a 255.

Cómo usarla:

1) Cargar el acumulador con el número de fichero lógico.
2) Cargar el registro X con el número de periférico.
3) Cargar el registro Y con la orden.

EJEMPLO:

```
FICHERO LOGICO 1, PERIFERICO 4 Y SIN COMANDO.
LDA #1
LDX #4
LDY #255
JSR SERTLFS
```

B-29. **Nombre de función: SETMSG**

Propósito: Controla la salida de los mensajes del sistema
Dirección de llamada: $FF90 (65424)
Registros de comunicación: A
Rutinas preparatorias: Ninguna
Errores devueltos: Ninguno
Necesidades de pila: 2
Registros afectados: A

Descripción: Esta rutina controla la impresión de los mensajes de control y error del KERNAL. Se puede seleccionar el mensaje de error mediante el acumulador. Un ejemplo de mensaje de error puede ser "FILE NOT FOUND". "PRESS PLAY ON TAPE" es un ejemplo de mensaje de control.
Los bits 6 y 7 de este valor determinan el tipo de mensaje que es. Si el bit 7 está a 1 es un mensaje de error. Si el bit 6 está a 1 es un mensaje de control.

Cómo usarla:

1) Coloque el valor deseado en el acumulador.
2) Llame a esta rutina.

EJEMPLO:

```
LDA #$40
JSR SETMSG          ;ACTIVA MENSAJES DE CONTROL
LDA #$80
JSR SETMSG          ;ACTIVA MENSAJES DE ERROR
LDA #0
JSR SETMSG          ;DESACTIVA TODOS LOS MENSAJES
```

B-30. Nombre de la función: SETNAM

Propósito: Ajustar el nombre del fichero
Dirección de llamada: $FFDB (65469)
Registros de comunicación: A, X, Y
Rutinas preparatorias: Ninguna
Necesidades de pila: Ninguna
Registros afectados: Ninguno

Descripción: Esta rutina se utiliza para ajustar el nombre del fichero para las rutinas OPEN, SAVE o LOAD. En el acumulador se debe cargar la longitud del nombre, y en los registros X e Y la dirección de inicio del nombre de fichero en el formato standard de byte bajo y byte alto. La dirección debe ser una posición de memoria válida donde debe encontrarse el nombre del fichero en ASCII. Si no se desea nombre de fichero, el acumulador debe estar a cero, representando una longitud de 0 caracteres. En este caso los registros X e Y pueden estar ajustados a cualquier posición de memoria.

Cómo usarla:

1) Cargue el acumulador con la longitud del nombre.
2) Cargue el registro X con el byte bajo de la dirección donde se encuentra el inicio del nombre.
3) Cargue el registro Y con el byte alto de la dirección mencionada.
4) Llame a esta rutina.

EJEMPLO:

```
LDA #NAME2-NAME     ;CARGA LA LONGITUD DEL NOMBRE
LDX #<NAME          ;CARGA LA DIRECCION DONDE SE ENCUENTRA
LDY #>NAME
JSR SETNAM
```


B-31. Nombre de función: SETTIM

Propósito: Ajustar el reloj del sistema
Dirección de llamada: $FFDB (65499)
Registros de comunicación: A, X, Y
Rutinas preparatorias: Ninguna
Errores devueltos: Ninguno
Necesidades de pila: 2
Registros afectados: Ninguno

Descripción: El reloj del sistema es mantenido por una de las rutinas de interrupción que lo actualiza cada 1/60 de segundo. El reloj consta de 3 bytes, lo cual le permite contar hasta 5.184.000 sesentaavos de segundo, lo que equivale a 24 horas. Al llegar a este límite se pone automáticamente a cero. Antes de llamar a esta rutina para poner en hora al reloj, el acumulador debe contener el byte más significativo, el registro X el siguiente y el registro Y el byte menos significativo.

Cómo usarla:

1) Cargue el acumulador con el byte más significativo de los tres necesarios para ajustar el reloj.
2) Cargue el registro X con el siguiente byte.
3) Cargue el registro Y con el byte menos significativo.
4) Llame a esta rutina.

EJEMPLO:

```
;AJUSTAR EL RELOJ A 10 MINUTOS (3600 SESENTAAVOS DE SEGUNDO)
LDA #0              ;MAS SIGNIFICATIVO
LDX #>3600
LDY #<3600          ;MENOS SIGNIFICATIVO
JSR SETTIM
```

B-32. Nombre de función: SETTMO

Propósito: Activar la bandera de espera del Bus IEEE
Dirección de llamada: $FFA2 (65422)
Registros de comunicación: A
Rutinas preparatorias: Ninguna
Errores devueltos: Ninguno
Necesidades de pila: 2
Registros afectados: Ninguno

Descripción: Esta rutina activa de bandera de espera del bus IEEE. Cuando esta bandera está activada, el Commodore 64 espera durante 64 milisegundos una señal del periférico conectado al Bus. Si el periférico no responde al Commodore 64 con la señal de validación de dirección de datos (DAV) en este tiempo, se produce una condición de error. Si se llama a esta rutina con un 0 en el bit 7 del acumulador se activa la bandera de espera. Si el bit 7 del acumulador está a 0, la bandera se desactiva.

NOTA: El Commodore 64 usa esta característica para comunicar que no se encuentra el fichero cuando se intenta OPEN únicamente con la tarjeta IEEE.

Cómo usarla:

PARA ACTIVAR LA BANDERA DE ESPERA:
1) Coloque el bit del acumulador a 0.
2) Llame a esta rutina.

PARA DESACTIVAR LA BANDERA DE ESPERA:
1) Coloque el bit 7 del acumulador a 1.
2) Llame a esta rutina.

EJEMPLO:

```
;DESACTIVAR LA BANDERA:
LDA #0
JSR SETTMO
```

B-33. Nombre de la función: STOP

Propósito: Comprobar si se ha pulsado la tecla STOP
Dirección de llamada: $FFE1 (65505)
Registros de comunicación: A
Rutinas preparatorias: Ninguna
Errores devueltos: Ninguno
Necesidades de pila: Ninguna
Registros afectados: A, X

Descripción: Si se pulsa la tecla STOP durante la ejecución de la rutina UDTIM, al llamar a la rutina STOP se activará la bandera Z. Además, se colocarán los valores standard en los canales. Las otras banderas quedarán inalteradas. Si no se ha pulsado la tecla STOP, el acumulador contendrá un byte que representa la última fila rastreada del teclado. Por este medio es posible también detectar si se han pulsado otras teclas.

Cómo usarla:

0) Debe llamar a UDTIM antes de utilizar la rutina.
1) Llame a esta rutina.
2) Compruebe la bandera de resultado cero (Z).

EJEMPLO:

```
JSR UDTIM       ;COMPRUEBA LA TECLA STOP
JSR STOP
ENE *+5         ;NO SE HA PULSADO
JMP READY       ;=... STOP
```


B-34. Nombre de la función: TALK

Propósito: Enviar el periférico del Bus serie orden de transmitir
Dirección de llamada: $FFB4 (65460)
Registros de comunicación: A
Rutinas preparatorias: Ninguna
Errores devueltos: Vea READST
Necesidades de pila: 8
Registros afectados: A

Descripción: El acumulador debe cargarse con el número de periférico (entre 0 y 31) antes de llamar a esta rutina. Cuando se llama, ejecuta un OR bit a bit para convertir este número en el orden TALK. Entonces se transmite este dato por el bus serie.

Cómo usarla:

1) Cargue el acumulador con el número de periférico.
2) Llame a esta rutina.

EJEMPLO:

```
;CONTROLAR EL PERIFERICO 4 PARA COMUNICACIONES
LDA #4
JSR TALK
```

B-35. Nombre de función: TKSA

Propósito: Enviar una dirección secundaria al periférico controlado por la rutina TALK
Dirección de llamada: $FF96 (65430)
Registros de comunicación: A
Rutinas preparatorias: TALK
Errores devueltos: Vea READST
Necesidades de pila: 8
Registros afectados: A

Descripción: Esta rutina envía una dirección secundaria a un periférico controlado por la rutina TALK. Esta rutina debe llamarse con el acumulador conteniendo un número entre 0 y 31. La rutina envia este número como dirección secundaria sobre el Bus Serie. Sólo puede llamarse a esta rutina después de TALK. No funciona después de LISTEN.

Cómo usarla:

0) Use la rutina TALK.
1) Cargue el acumulador con la dirección secundaria.
2) Llame a esta rutina.

EJEMPLO:

```
;ENVIA LA DIRECCION SECUNDARIA 7 AL PERIFERICO 4
LDA #4
JSR TALK
LDA #7
JSR TKSA
```

B-36. Nombre de función: UDTIM

Propósito: Actualizar el reloj del sistema
Dirección de llamada: $FFEA (65514)
Registros de comunicación: Ninguno
Rutinas preparatorias: Ninguna
Errores devueltos: Ninguno
Necesidades de pila: 2
Registros afectados: A, X

Descripción: Esta rutina actualiza el reloj del sistema. Normalmente la utiliza la rutina de interrupción del KERNAL cada 1/60 de segundo. Si el programa del usuario interfiere las rutinas normales de interrupción, debe utilizarse esta rutina para actualizar el tiempo. Igualmente debe llamarse a la rutina STOP si se desea que esta tecla funcione.

Cómo usarla:

1) Llamar a esta rutina.

EJEMPLO:

```
JSR UDTIM
```

B-37. Nombre de función: UNLSN

Dirección de llamada: $FFAE (65454)
Registros de comunicación: Ninguno
Rutinas preparatorias: Ninguna
Errores devueltos: Vea READST
Necesidades de pila: 8
Registros afectados: A

Descripción: Esta rutina ordena a todos los periféricos del Bus Serie que detengan la recepción de datos desde el Commodore 64. LLamando a esta rutina se envía una orden UNLISTEN sobre el Bus Serie. Sólo quedan afectados los periféricos a los que anteriormente se les envió la orden LISTEN. Normalmente se llama a esta rutina cuando el Commodore 64 ha finalizado el envío de información a los periféricos. Enviándola se recuperan los periféricos para utilizarlos en otras funciones.

Cómo usarla:

1) Llamar a esta rutina.

EJEMPLO:

```
JSR UNLSN
```

B-38. Nombre de la función: UNTLK

Propósito: Enviar la orden UNTALK
Dirección de llamada: $FFAB (65451)
Registros de comunicación: Ninguno
Rutinas preparatorias: Ninguna
Errores devueltos: Vea READST
Necesidades de pila: 8
Registros afectados: A

Descripción: Esta rutina transmite la orden UNTALK al Bus Serie. Todos los periféricos afectados por la rutina TALK detendrán el envío de datos cuando reciban esta orden.

Como usarla:

1) LLamar a esta rutina

EJEMPLO:

```
JSR UNTLK
```

B-39. Nombre de función: VECTOR

Propósito: Manejo de los vectores de RAM
Dirección de llamada: $FF8D (65421)
Registros de comunicación: X, Y
Rutinas preparatorias: Ninguna
Errores devueltos: Ninguno
Necesidades de pila: 2
Registros afectados: A, X, Y

Descripción: Esta rutina maneja todos los vectores del sistema almacenados en RAM. Llamando a esta rutina con el bit de acarreo del acumulador activado, se almacena el contenido actual de los vectores en RAM en una lista apuntada por los registros X e Y. Cuando esta rutina es llamada con el acarreo desactivado, la lista de vectores confeccionada por el usuario y punteada por los registros X e Y es transferida a la lista de vectores en RAM del sistema.

> **NOTA:** Esta rutina precisa cuidado en su uso. La mejor manera de utilizarla es leer primero el contenido entero de los vectores en el área del usuario, alterar los vectores deseados y copiar la nueva configuración en los vectores del sistema.

Cómo usarla:

LEER LOS VECTORES DEL SISTEMA EN RAM:

1) Activar el acarreo.
2) Colocar en los registros X e Y la dirección de la zona donde se desean tener los vectores.
3) Llamar a esta rutina.

COLOCAR LOS VAORES MODIFICADOS EN LOS VECTORES DEL SISTEMA EN RAM

1) Desactivar el acarreo.
2) Ajustar los registros X e Y a la dirección donde se inicia la tabla de valores que debe ser cargada en los vectores.
3) Llamar a esta rutina.

EJEMPLO:

```
;CAMBIAR LAS  RUTINAS DE INPUT UN NUEVO SISTEMA.
LDX #<USER
LDY #>USER
SEC
JSR VECTOR      ;LEE LOS ANTIGUOS VALORES
LDA #<MYINP     ;CAMBIA EL VECTOR DE INPUT
STA USER+10
LDA #MYINP
STA USER+11
LDX #<USER
LDY #>USER
CLC
JSR VECTOR      ;ALTERA EL SISTEMA
...
USER *=*+26
```

CODIGOS DE ERROR

A continuación se muestra una lista de mensajes de error que se pueden producir al utilizar las rutinas de KERNAL. Si se produce un error, el bit de acarreo del acumulador se activa, y se coloca el número del mensaje de error en el acumulador.

> NOTA: Algunas rutinas del KERNAL no utilizan estos códigos de error. En su lugar, los errores se identifican mediante la rutina READST.

NUMERO	SIGNIFICADO
0	Rutina finalizada por la tecla STOP.
1	Demasiados ficheros abiertos.
2	Fichero abierto previamente.
3	Fichero no abierto.
4	No se ha encontrado el fichero.
5	Periférico no conectado.
6	El fichero no es de entrada.
7	El fichero no es de salida.
8	Falta el nombre del fichero.
9	Número de periférico ilegal.
240	Cambio del final de memoria para alojar/desalojar el buffer del RS-232.

USO DEL LENGUAJE MAQUINA DESDE EL BASIC

Hay varios métodos disponibles en su Commodore 64 para usar conjuntamente BASIC y lenguaje máquina, incluyendo instrucciones especiales del BASIC, así como posiciones clave en el sistema operativo. Existen cinco formas distintas de utilizar rutinas en lenguaje máquina desde el BASIC:

1) La instrucción SYS del BASIC.
2) La instrucción USR del BASIC.
3) Cambiando uno de los vectores de E/S en RAM.
4) Cambiando uno de los vectores de interrupción en RAM.
5) Cambiando la rutina CHRGET.

1) La instrucción **SYS X** del BASIC causa un **SALTO** a una rutina en lenguaje máquina colocada a partir de la dirección X. La rutina debe finalizar con la instrucción **RTS** (Volver de subrutina). De esta forma se transfiere nuevamente el control al **BASIC**.
   Los parámetros se pasan entre la rutina en lenguaje máquina y el BASIC mediante las instrucciones PEEK y POKE del BASIC, y sus equivalentes en lenguaje máquina.
   La instrucción **SYS** es el método más simple y útil de combinar BASIC, y sus equivalentes en lenguaje máquina. Los PEEKs y POKEs permiten pasar fácilmente los parámetros necesarios. Pueden existir varias instrucciones SYS en un programa, cada una refiriéndose a una rutina distinta, o refiriéndose todas a la misma.

2) La función **USR(X)** del BASIC transfiere el control a una subrutina en lenguaje máquina cuya dirección se encuentra almacenada en las posiciones 785 y 786. (La dirección se almacena en el formato standard byte bajo/byte alto). El valor de

X es evaluado y pasado a la rutina en lenguaje máquina a través del acumulador en coma flotante #1, cuyo inicio se encuentra en la posición $61 (consulte el mapa de memoria para más detalles). Puede ser devuelto un valor al BASIC si se coloca en dicho acumulador. La rutina en lenguaje máquina debe terminarse con la instrucción RTS para devolver el control al BASIC.
Esta instrucción difiere de la **SYS** en que debe ajustar un vector indirecto. El vector indirecto es el formato mediante el cual se pasa la variable de coma flotante a la rutina en C/M. Es necesario cambiar el vector si se precisa más de una rutina en lenguaje máquina.

3) Cualquiera de las rutinas internas del BASIC a las que se accede mediante los vectores de E/S situados en la página 3 (Vea **MODOS DE DIRECCIONAMIENTO, PAGINA CERO**) puede ser modificada o reemplazada por la rutina propia del programador. Cada vector de 2 bytes consiste en la dirección (byte/bajo/byte alto) utilizada por el sistema operativo.
La rutina **VECTOR DEL KERNAL** es la forma más sencilla de cambiar cualquiera de los vectores, pero si sólo se desea cambiar uno, se puede hacer mediante POKE o su equivalente en lenguaje máquina. El nuevo vector debe dirigirse a la dirección de inicio de la rutina utilizada para cambiar o potenciar la rutina original del sistema. Cuando se ejecuta el comando BASIC afectado, se ejecutará la rutina del usuario. Si después de ejecutada ésta es necesario ejecutar también la rutina del sistema, el programa del usuario debe terminar con la instrucción **JMP** (salto) a la dirección original del vector. Si no es así, la rutina debe terminar con RTS para devolver el control al BASIC.

4) El **VECTOR DE INTERRUPCION DEL HARDWARE (IRQ)** puede ser cambiado. Cada sesentaavo de segundo, el sistema operativo transfiere el control a la rutina especificada en este vector. El KERNAL utiliza normalmente esto para actualizar el reloj, rastrear el teclado, etc. Si se usa esta técnica, debe siempre transferir el control a la rutina normal de IRQ a menos que la rutina que la reemplace esté preparada para manejar el chip CIA. (RECUERDE terminar su rutina con la instrucción RTI si la CIA es manejada por su rutina).
Este método es útil para tareas que deban funcionar al mismo tiempo que un programa en BASIC, pero tiene el inconveniente de ser más difícil.

NOTA: ¡¡¡DESACTIVE SIEMPRE LAS INTERRUPCIONES ANTES DE CAMBIAR ESTE VECTOR!!!

5) La rutina **CHRGET** se usa por el BASC para encontrar cada carácter/token (token = código de un byte en que se transforma cada instrucción BASIC). Esto permite fácilmente añadir nuevos comandos BASIC. Naturalmente, cada nuevo comando debe ser escrito en lenguaje máquina. Una forma común de usar este método es especificar un caracter (@ por ejemplo) para que realice una nueva tarea. La nueva rutina CHRGET busca este carácter especial. Si no se encuentra, el control se pasa a la rutina CHRGET normal del BASIC. Si el carácter especial se localiza, el nuevo comando se interpreta y ejecuta por su rutina en lenguaje máquina. Esto minimiza el tiempo extra de ejecución preciso para buscar los nuevos comandos. Esta técnica se suele llamar "cuna".

DONDE COLOCAR RUTINAS EN LENGUAJE MAQUINA

El mejor lugar para colocar rutinas en C/M en el Commodore 64 es en la zona entre $C000–$CFFF, asumiendo que éstas sean menores de 4K bytes. Esta sección de memoria está libre y no es perturbada por el BASIC.
Si por alguna razón no es posible o deseable colocar la rutina en lenguaje máquina en $C000, por ejemplo si la rutina ocupa más de 4K., es necesario reservar una zona de memoria para la rutina, de forma que el BASIC no pueda acceder a ella. El final de memoria se sitúa normalmente en $9FFF. Este final puede ser cambiado mediante la rutina del KERNAL MEMTOP, o mediante las siguientes instrucciones BASIC:

```
10 POKE51,L:POKE52,H:POKE55,L:POKE56,H:CLR
```

Donde H y L son los bytes alto y bajo respectivamente de la nueva dirección de final de memoria. Por ejemplo, si desea reservar un área desde $9000 hasta $9FFF para sus programas en lenguaje máquina, utilice lo siguiente:

```
10 POKE51,0:POKE52,144:POKE55,0:POKE56,144:CLR
```

COMO ENTRAR PROGRAMAS EN LENGUAJE MAQUINA

Existen 3 métodos comunes de entrar programas en el **lenguaje máquina:**

1) INSTRUCCIONES DATA:

Leyendo instrucciones DATA (READ) y colocando sus valores en la zona de memoria deseada (POKE) se colocará en memoria una rutina el lenguaje máquina en memoria. Este es el método mas simple. No se precisan métodos especiales para guardar el programa y es sencillo depurarlo. Los inconvenientes son que ocupa más espacio y se debe esperar mientras se coloca en memoria. Sin embargo, este método es muy útil para rutinas cortas.

EJEMPLO:

```
10 RESTORE:FOR X=1TO9:READA:POKE12*4096+X,A:NEXT
.
.
.
PROGRAMA EN BASIC
.
.
.
1000 DATA 161,1,204,204,204,204,204,204,96
```

2) MONITOR DE LENGUAJE MAQUINA(64MON)

Este programa le permite entrar sus programa en C/M en **hexadecimal o en sus** mnemónicos, y grabar la parte de memoria donde se almacena el programa. **Las** ventajas de este método incluyen una gran facilidad **en la introducción del**

programa, corrección y facilidad de guardar y cargar programas. El inconveniente es que normalmente será necesario un programa en BASIC para cargar la rutina en lenguaje máquina. (Para más detalles sobre el 64MON vea la sección de programación en lenguaje máquina).

EJEMPLO

A continuación se muestra un típico programa en BASIC que carga un programa en C/M grabado con el 64MON:

```
10 IF FLAG=1 THEN 20
15 FLAG=1:LOAD "NOMBRE DE LA RUTINA EN C/M",1,1
20
```

RESTO DEL PROGRAMA EN BASIC

3) PAQUETE ENSAMBLADOR/EDITOR

Las ventajas son similares al uso de un monitor en lenguaje máquina, pero los programas son incluso más fáciles de entrar. Los inconvenientes son los mismos que los reseñados para el 64MON.

MAPA DE MEMORIA DEL COMMODORE 64

ETIQUETA	HEX	DECIMAL	DESCRIPCION
D6510	0000	0	Registro dirección de datos 6510.
R6510	0001	1	Registro E/S 8 bit 6510.
	0002	2	Sin uso.
ADRAY1	0003–0004	3–4	Vector Flotante-Fijo.
ADRAY2	0005–0006	5–6	Vector Fijo-Flotante.
CHARAC	0007	7	Busca carácter.
ENDCHR	0008	8	Bandera búsqueda de comillas.
TRMPOS	0009	9	Guarda la columna de TAB.
VERCK	000A	10	0=LOAD, 1=VERIFY.
COUNT	000B	11	Puntero del Buffer de entr./Núm. de suscritos.
DIMFLG	000C	12	Bandera: Dimensión normal de Tablas.
VALTYP	000D	13	Tipo de datos: $FF=cadena, $00=Numérico.
INTFLG	000E	14	Tipo de datos: $80=Enteros, $00=Flotantes.
GARBFL	000F	15	Bandera: Busca DATA/comillas LIST/GarbageCollection.
SUBFLG	0010	16	Bandera: Llamada a FN.

ETIQUETA	HEX	DECIMAL	DESCRIPCION
INPFLG	0011	17	Bandera: $00=INPUT, $40=GET, $98=READ.
TANSGN	0012	18	Bandera: Signo TAN/Resultado de la comparación.
	0013	19	Bandera: Comillas INPUT.
LINNUM	0014–0015	20–21	Valor entero..
TEMPPT	0016	22	Puntero: Stack temporal de cadenas.
LASTPT	0017–0018	23–24	Vector de último literal.
TEMPST	0019–0021	25–33	Pila temporal de literales.
INDEX	0022–0025	34–37	Area de punteros de utilidad.
RESHO	0026–002A	38–42	Producto de multiplicación Coma flotante.
TXTTAB	002B–002C	43–44	Puntero: Inicio del BASIC.
VARTAB	002D–002E	45–46	Puntero: Inicio de variables.
ARYTAB	002F–0030	47–48	Puntero: Inicio de tablas.
STREND	0031–0032	49–50	Puntero: Fin de tablas (+1).
FRETOP	0033–0034	51–52	Puntero: Inicio de cadenas.
FRESPC	0035–0036	53–54	Puntero de cadenas, Utilidad.
MEMSIZ	0037–0038	55–56	Puntero: Máxima dirección utilizable por el BASIC.
CURLIN	0039–003A	57–58	Número de línea actual BASIC
OLDLIN	003B–003C	59–60	Número de línea previo.
OLDTXT	003D–003E	61–62	Puntero de dirección para CONT.
DATLIN	003F–0040	63–64	Número de línea DATA actual.
DATPTR	0041–0042	65–66	Puntero: Dirección del actual ítem de DATA.
INPPTR	0043–0044	67–68	Vector: Rutina de INPUT.
VARNAM	0045–0046	69–70	Actual nombre de variable BASIC.
VARPNT	0047–0048	71–72	Puntero: Dato actual de la variable BASIC.
FORPNT	0049–004A	73–74	Puntero: Variable de índice FOR-NEXT.
	004B–0060	75–96	Punteros temporales del área de datos.
FACEXP	0061	97	Exponente del acumulador de coma flotante 1.
FACHO	0062–0065	98–101	Mantisa acum. 1.
FACSGN	0066	102	Signo Acum. 1.
SGNFLG	0067	103	Puntero: Constante de evaluación de series.
BITS	0068	104	Acum. 1. Dígito de overflow.
ARGEXP	0069	105	Acum. 1: Exponente.
ARGHO	006A–006D	106–109	Acum. 2: Mantisa.

ETIQUETA	HEX	DECIMAL	DESCRIPCION
ARGSGN	006E	110	Acum. 2. Signo.
ARISGN	006F	111	Signo del resultado de la Comparación entre el Acum. 1 y el Acum. 2.
FACOV	0070	112	Acum. #1: redondeo.
FBUFPT	0071–0072	113–114	Puntero: Buffer del Cassette.
CHRGET	0073–008A	115–138	Subrutina: Busca el próximo carácter del BASIC.
CHRGOT	0079	121	Entrada para buscar otra vez el mismo byte de texto.
TXTPTR	007A–007B	122–123	Puntero: Byte actual del texto BASIC.
RNDX	008B–008F	139–143	Valor de la semilla de la función RND.
STATUS	0090	144	Palabra de estado del KERNAL para E/S (ST).
STKEY	0091	145	Bandera: Teclas STOP/RVS.
SVXT	0092	146	Constante de tiempo del Cass.
VERCK	0093	147	Bandera: 0=LOAD, 1=VERIFY.
C3P0	0094	148	Bandera: Salida de carácter por el Bus Serie
BSOUR	0095	149	Carácter con buffer para el Bus Serie
SYNO	0096	150	Núm. Sincronización Cassette.
LDTNC	0097	151	Area de datos temporales.
DFLTN	0098	152	Núm. ficheros abiertos/Indice a la tabla de ficheros.
DFLTN	0099	153	Periférico de entrada normal (0).
DFLTO	009A	154	Periférico normal de salida (CMD) (3).
PRTY	009B	155	Paridad de carácter (casset).
DPSW	009C	156	Bandera: Byte recibido (cass).
MSGFLG	009D	157	Bandera: $80=Modo directo, $00=Modo programa.
PTR1	009E	158	Error paso de cinta 1.
PTR2	009F	159	Error paso de cinta 2.
TIME	00A0–00A2	160–162	Reloj tiempo real (1/60 seg).
	00A3–00A4	163–164	Area de datos temporales.
CNTDN	00A5	165	Contador de sincronización del Cassette.
BUFPNT	00A6	166	Puntero: Buffer E/S del cass
INBIT	00A7	167	RS-232 Bits de input/Temp de cassette.
BITCI	00A8	168	RS-232 Cuenta bit de input/ Temp. cassette.

ETIQUETA	HEX	DECIMAL	DESCRIPCION
RINONE	00A9	169	Bandera RS-232: Comprueba el Bit de Start.
RIDATA	00AA	170	RS-232 Byte de input buffer/ Temp. de cassette.
RIPRTY	00AB	171	RS-232 Paridad/corto CNT de cassette.
SAL	00AC–00AD	172–173	Puntero: Buffer cassette/ scroll de pantalla.
EAL	00AE–00AF	174–175	Direcciones de fin de cinta/ fin de programa.
CMPO	00B0–00B1	176–177	Constantes de cinta, tiempos.
TAPE1	00B2–00B3	178–179	Punter: Inicio del buffer Puntero del Cassette.
BITTS	00B4	180	RS-232 cuenta bit de salida Tem. Cassette.
NXTBIT	00B5	181	RS-232 Próximo bit a enviar/ Bandera de fin de cinta.
RODATA	00B6	182	Rs-232 Buffer de salida.
FNLEN	00B7	183	Longitud del nombre de fichero actual.
LA	00B8	184	Número de fichero lógico actual.
SA	00B9	185	Dirección secundaria actual.
FA	00BA	186	Número de periférico actual.
FNADR	00BB–00BC	187–188	Puntero: Nombre del fichero actual.
ROPRTY	00BD	189	RS-232 paridad salida/ Temp. cassette.
FSBLK	00BE	190	Contador de bloques R/W cass.
MYCH	00BF	191	Buffer de palabra serie.
CAS1	00C0	192	Trabado del motor del casset.
STAL	00C1–00C2	193–194	Dirección de inicio E/S.
MEMUSS	00C3–00C4	195–196	Tem de LOAD cassette.
LSTX	00C5	197	Tecla pulsada: CHR$(n). 0= no hay tecla pulsada.
NDX	00C6	198	Núm. de caracteres en el buffer de teclado (cola).
RVS	00C7	199	Bandera: Imprimir caracteres inversos. 1=Sí/0=No.
INDX	00C8	200	Puntero: Fin de línea lógica para INPUT.
LXSP	00C9–00CA	201–202	Pos X,Y del cursor al inicio de INPUT.
SFDX	00CB	203	Bandera: Imprimir caracteres con Shift.
BLNSW	00CC	204	Parpadeo del cursor: 0=Sí.

ETIQUETA	HEX	DECIMAL	DESCRIPCION
BLNCT	00CD	205	Timer: Tiempo entre parpadeo del cursor.
GDBLN	00CE	206	Carácter bajo el cursor.
BLNON	00CF	207	Bandera: último parpadeo del cursor. Encend./Apag.
CRSW	00D0	208	Bandera: INPUT o GET desde el teclado.
PNT	00D1–00D2	209–210	Puntero: Dirección actual de la línea de pantalla.
PNTR	00D3	211	Columna del cursor en la línea actual.
QTSW	00D4	212	Bandera: Editor en modo comillas. 0=NO.
LNMX	00D5	213	Longitud de la línea física.
TBLX	00D6	214	Pos. del cursor en la línea física.
	00D7	215	Area de datos Temp.
INSRT	00D8	216	Bandera: Modo insert.>Ø=NO. IMSERC.
LDTB1	00D9–00F2	217–242	Tabla de link de pantalla/Temp. editor.
USER	00F3–00F4	243–244	Puntero de color de pantalla.
KEYTAB	00F5–00F6	245–246	Vector: Tabla de decodificación del teclado.
RIBUF	00F7–00F8	247–248	Puntero del buffer de entrada RS–232.
ROBUF	00F9–00FA	249–250	Puntero del buffer de salida del RS-232.
FREKZP	00FB–00FE	251–254	Espacio libre en pág. 0 para necesidades de programación.
BASZPT	00FF	255	Area de datos Temp BASIC.
	0100–01FF	256–511	Area de Stack del procesador.
	0100–010A	256–266	Area de trabajo de cadenas.
BAD	0100–013E	256–318	Entrada de error cassette.
BUF	0200–0258	512–600	Buffer INPUT del sistema.
LAT	0259–0262	601–610	Tabla KERNAL: Núm. de fichero lógico activo.
FAT	0263–026C	611–620	Tabla KERNAL: Núm. periférico para cada fichero.
SAT	026D–0276	621–630	Tabla KERNAL: Dirección secundaria de cada fichero.
KEYDE	0277–0280	631–640	Buffer de teclado (FIFO).
MEMSTR	0281–0282	641–642	Puntero: Principio memoria para el O.S.
MEMSIZ	0283–0284	643–644	Puntero: Fin de memoria para el O.S.

ETIQUETA	HEX	DECIMAL	DESCRIPCION
TIMOUT	0285	645	Bandera: variable KERNAL para espera IEEE.
COLOR	0286	646	Código de color actual.
GDCOL	0287	647	Color de fondo bajo el cursor.
HIBASE	0288	648	Fin de memoria de pantalla (página)
XMAX	0289	649	Tamaño del Buffer de teclado.
RPTFLG	028A	650	Bandera: repetición de tecla $80=repetición
KOUNT	028B	651	Contador de velocidad de repetición
DELAY	028C	652	Contador de retraso en la repetición
SHFLAG	028D	653	Bandera tecla Shift/CTRL/C<
LSTSHF	028E	654	Modelo de la última mayúscu.
KEYLOG	028F–0290	655–656	Vector: activación de la tabla de teclado
MODE	0291	657	Bandera: $00=desactivar teclas SHIFT/$80=activarlas
AUTODN	0292	658	Bandera: Scroll automático 0=SI
M51CTR	0283	659	RS–232: registro de control de imagen del 6551
M51CDR	0294	660	RS–232: registro de comando de imagen del 6551
M51AJB	0295–0296	661–662	RS–232 no Standard (USA) PBS (TIEMPO/2–100)
RSSTAT	0297	663	RS–232 Registro de estado de 6561
BITNUM	0298	664	RS–232 Número de bits a env.
BAUDOF	0299–029A	665–666	RS–232 baudios (tiempo) TIEMPO BIT (µs).
RIDBE	029B	667	Indice de fin debuffer de entrada del RS–232.
RIDBS	029C	668	Página de inicio del buffer de entrada RS–232.
RODBS	029D	669	Página de inicio del buffer de salida RS–232.
RODBE	029E	670	Indice de fin de buffer de salida del RS–232.
IRQTMP	029F–02A0	671–672	Almacena el vector de IRQ durante las E/S con el cass.
ENABL	02A1	673	Activa RS–232.

ETIQUETA	HEX	DECIMAL	DESCRIPCION
	02A2	674	TOD durante E/S del cassette
	02A3	675	Almacenamiento Temp. para lectura del cassette.
	02A4	676	Indicador D1IRQ durante lectura del cassette.
	02A5	677	Indice de línea.
	02A6	678	Bandera: 0=NTSC/1=PAL.
	02A7–02FF	679–767	No usado.
IERROR	0300–0301	768–769	Vector: imprime mensajes de error BASIC.
IMAIN	0302–0303	770–771	Vector: inicio de BASIC.
ICRNCH	0304–0305	772–773	Vector: Conversión de texto BASIC.
IQPLOP	0306–0307	774–775	Vector: Listados BASIC.
IGONE	0308–0309	776–777	Vector: enlaces BASIC CAR.
IEVAL	030A–030B	778–779	Vector: evaluación de código BASIC.
SAREG	030B	780	Almacenamiento del registro A del 6510.
SXREG	030C	781	Almacenamiento del registro X del 6510.
SYREG	030E	782	Almacenamiento del registro Y del 6510.
SPREG	030F	783	Almacenamiento del registro SP del 6510.
USRPOK	0310	784	Instrucción JMP para USR.
USRADD	0311–0312	785–786	Dirección USR (bajo/alto).
	0313	787	Sin uso.
CINV	0314–0315	788–789	Vector: interrupción del hard ware (IRQ).
CBINV	0316–0317	790–791	Vector: interrupción por BRK.
NMINV	0318–0319	792–793	Vector: interrupción no enmascarable
IOPEN	031A–031B	794–795	RUTINA OPEN
ICLOSE	031C–031D	796–797	RUTINA CLOSE
ICHKIN	031E–031F	798–799	RUTINA CHKIN
ICKOUT	0320–0321	800–801	RUTINA CHKOUT
[illegible]RCH	0322–0323	802–803	RUTINA CLRCHN
[illegible]IN	0324–0325	804–805	RUTINA CHRIN

ETIQUETA	HEX	DECIMAL	DESCRIPCION
IBSOUT	0326–0327	806–807	RUTINA CHROUT
ISTOP	0328–0329	808–809	RUTINA STIOP
IGETIN	032A–032B	810–811	RUTINA GETIN
ICLALL	032C–032D	812–813	RUTINA CLALL
USRCMD	032E–032F	814–815	RUTINA DEFINIBLE USUARIO
ILOAD	0330–0331	816–817	RUTINA LOAD
ISAVE	0332–0333	818–819	RUTINA SAVE
	0334–033B	820–1019	Sin uso
TBUFFR	033C–03FB	828–1019	Buffer de E/S del cassette
	03FC–03FF	1020–1023	Sin uso
VICSCN	0400–07FF	1024–2047	Area de 1024 bytes de memoria de pantalla.
	0400–07E7	1024–2023	Matriz de vídeo: 25 líneas 40 columnas.
	07F8–07FF	2040–2047	Punteros de Sprites
	0800–9FFF	2048–40959	Espacio normal para programas en BASIC.
	8000–9FFF	32768–40959	Cartucho de ROM.8192 bytes.
	A000–BFFF	40960–49151	ROM del BASIC. 8192 bytes (u 8K de RAM).
	C000–CFFF	49152–53247	RAM–4096 bytes.
	D000–DFFF	53248—57343	RAM de color y de E/S o juego de caracteres ROM o RAM (4096 bytes).
	E000–FFFF	57344–65535	ROM del KERNAL–8192 bytes (u 8K. de RAM).

ENTRADA/SALIDA DEL COMMODORE 64

HEX	DECIMAL	BITS	DESCRIPCION
0000	0	7–0	Registro de dirección de datos del 6510 (xx101111) Bit=1:Salida, Bit=0:Entrada x=sin cuidado.
0001	1		Port de E/S del 6510 LORAM (0=BASIC desconectado).
		1	HIRAM (0=Desconectada el KERNAL).
		2	CHAREN (0=conectada ROM de caracteres).

HEX	DECIMAL	BITS	DESCRIPCION
		3	Línea de salida de datos del cassette.
		4	Interruptor de sentido del cassette. 1=cerrado.
		5	Control del motor del cassette 0=funciona 1=no funciona
		6-7	Indefinidos.
D000-D02E	53248-53294		**CONTROLADOR DE VIDEO MOS 6566 (VIC-II).**
D000	53248		Pos X Sprite 0.
D001	53249		Pos Y Sprite 0.
D002	53250		Pos X Sprite 1.
D003	53251		Pos Y Sprite 1.
D004	53252		Pos X Sprite 2.
D005	53253		Pos Y Sprite 2.
D006	53254		Pos X Sprite 3.
D007	53255		Pos Y Sprite 3.
D008	53256		Pos X Sprite 4.
D009	53257		Pos Y Sprite 4.
D00A	53258		Pos X Sprite 5.
D00B	53259		Pos Y Sprite 5.
D00C	53260		Pos X Sprite 6.
D00D	53261		Pos Y Sprite 6.
D00E	53262		Pos X Sprite 7.
D00F	53263		Pos Y Sprite 7.
D010	53264		MSB de la posición X de los Sprites 0-7.
D011	53265		Registro de control del VIC-II.
		7	Comparación de barrido (bit8). Vea 53266.
		6	Modo de color de texto extendido. 1=activado.
		5	Modo Bit-Map. 1=Activado.
		4	Pantalla a color de borde. 1=Activado Ø=Pantalla Eliminada.
		3	Selec. Display de 24/25 líneas. 1=25 líneas.
		2-0	Scroll uniforme a la posic. Y (0-7).
D012	53266		Valor de L/E de barrido para comparar en IRQ.
D013	53267		Pos X de lápiz óptico.
D014	53268		Pos Y de lápiz óptico.
D015	53269		Activación de Sprites. 1=act.
D016	53270		Registro de control del VIC-II.

HEX	DECIMAL	BITS	DESCRIPCION
		7-6	Sin uso.
		5	MANTENGA SIEMPRE ESTE BIT A OI.
		4	Modo Multicolor. 1=activado. (Texto o Bit Map).
		3	Seleccionar 38/49 columnas 1=40 columnas.
		2-0	Scroll uniforme a la posic. X.
D017	53271		Expandir Sprites 0-7 en vertical (Y) 2X.
D018	53272		Registro de control de memoria del VIC-II.
		7-4	Dirección de base de la matriz de vídeo.
		3-1	Dirección de base de los datos sobre caracteres.
D019	53273		Bandera del registro de IRQ. (Bit=1:Hay IRQ).
		7	Activado en cualquier condición de IRQ.
		3	Bandera de IRQ de lápiz óptico.
		2	Bandera IRQ de colisión entre Sprites.
		1	Bandera IRQ de colisión entre Sprites y datos.
		0	Bandera IRQ de comparación de barrido.
D01A	53274		Registro de máscara de IRQ. 1=interrupción activada.
D01B	53275		Prioridad de Sprites y/o caracteres. 1=Sprite.
D01C	53276		Modo Multicolor de Sprites 0-7: 1=Modo activado.
D01D	53277		Expansión horizontal (X) de los Sprites 0-7 (2X).
D01E	53278		Detección de colisiones entre Sprites.
D01F	53279		Detección de colisiones entre Sprites y datos.
D020	53280		Color del borde.
D021	53281		Color de fondo 0.
D022	53282		Color de fondo 1.
D023	53283		Color de fondo 2.
D024	53284		Color de fondo 3.
D025	53285		Registro de Sprite multicolor. 0.

HEX	DECIMAL	BITS	DESCRIPCION
[illegible]	53286		Registro de Sprite multicolor. 1
D027	53287		Color de Sprite 0.
[illegible]	53288		Color de Sprite 1.
D029	53289		Color de Sprite 2.
[illegible]	53290		Color de Sprite 3.
[illegible]	53291		Color de Sprite 4.
D02C	53292		Color de Sprite 5.
D02D	53293		Color de Sprite 6.
D02F	53294		Color de Sprite 7.
D400–D7FF	54272–55295		CHIP DE SONIDO MOS 6581 (SID)
D400	54272		Voz 1: Byte bajo de frecuencia.
D401	54273		Voz 1: Byte alto de frecuencia.
D402	54274		Voz 1: Anchura de pulso (Bajo).
D403	54275	7–4	Sin uso
		3–0	Voz 1: Anchura de pulso (Alto).
D404	54276	7	Onda aleatoria de ruido. 1=Sí.
		6	Onda de pulso, 1=Sí.
		5	Onda diente de sierra, 1=Sí.
		4	Onda triangular, 1=Sí.
		3	Test bit, 1=desactiva oscil 1
		2	Modulación en anillo. Osc. 1 con salida del osc.3. 1=activado.
		1	Sincronización Osc.1 con 3. 1=activado.
		0	Bit de puerta. 1=empezar Ataq/Deca/Sos. 0=empezar relajación
D405	54277		Generador de envolvente Osc. 1.
		7–4	Duración Ataque (0–15).
		3–0	Duración Decaimiento (0–15).
D406	54278		Generador de envolvente Osc. 1.
		7–4	Ciclo de Sostenimiento (0–15).
		3–0	Duración de relajación (0–15).
D407	54279		Voz 2. Frecuencia (bajo).
D408	54280		Voz 2. Frecuencia (alto).
D409	54281		Voz 2. Anchura de pulso (bajo).
D40A	54282	7–4	Sin uso.

HEX	DECIMAL	BITS	DESCRIPCION
		3–0	Voz 2. Anchura de pulso (alto).
D40B	54283		Registro de control voz 2.
		7	Onda de ruido. 1=Sí.
		6	Onda de pulso. 1=Sí.
		5	Diente de sierra. 1=Sí.
		4	Onda triangular. 1=Sí.
		3	1=Desactiva Osc. 2.
		2	Modulación de timbre Osc. 2 con salida Osc. 1 1=activado.
		1	Sincronización Osc. 2 con 1. 1=activado.
		0	1=Bit de puerta. 0=Inicio Relajación.
D40C	54284		Generador de envolvente 2.
		7–4	Ciclo de Ataque (0–15).
		3–0	Ciclo de decaimiento. (0–15).
D40D	54285		Generador de envolvente 2.
		7–4	Ciclo de sostenimiento (0–15).
		3–0	Ciclo de relajación (0–15).
D40E	54286		Voz 3. Frecuencia (bajo).
D40F	54287		Voz 3. Frecuencia (alto).
D410	54288		Voz 3. Anchura de pulso (bajo).
D411	54289	7–4	Sín uso.
		3–0	Anchura de pulso (alto).
D412	54290		Registro de control voz 3.
		7	Onda de ruido. 1=Sí.
		6	Onda de pulso. 1=Sí.
		5	Diente de Sierra. 1=Sí.
		4	Onda triangular. 1=Sí.
		3	1=Desactiva oscilador 3.
		2	Modulación en anillo entre Osc. 3 y 2. 1=Sí.
		1	Sincronización entre Osc. 3 Osc. 2. 1=Sí.
		0	1=Alta/Dec/Sost, 0=Relajación.
D413	54291		Generador de envolvente 3.
		7–4	Ataque (0–15).
		3–0	Decaimiento (0–15).
D414	54292		Generador de envolvente 3.
		7–4	Sostenimiento (0–15).
		3–0	Relajación (0–15).
D415	54293		Frecuencia de corte del filtro (bajo) (Bits 2–0).

HEX	DECIMAL	BITS	DESCRIPCION
D416	54294		Byte alto de la frecuencia de corte del filtro.
D417	54295		Control de resonancia del filtro/Control de voz de entrada.
		7–4	Selección de resonancia del filtro (0–15).
		3	Filtro de entrada externa 1=SI, 0=NO.
		2	Salida filtrada de voz 3. 1=SI, 0=NO.
		1	Salida filtrada de voz 2. 1=SI, 0=NO.
		0	Salida filtrada de voz 1. 1=SI, 0=NO.
D418	54296		Selección de filtro y volumen.
		7	Corte de la salida de la voz 3. 1=Corte. 0=No corte.
		6	Selección de filtro pasaaltos 1=Sí.
		5	Selección de filtro pasabanda. 1=Sí.
		4	Selección filtro pasabajos. 1=SI.
		3–0	Selección de volumen de salida (0–15).
D419	54297		Conversor A/D Paddle 1 (0–255).
D41A	54298		Conversor A/D Paddle 2 (0–255).
D41B	54299		Generador números aleatorios. Osc.3
D41C	54300		Salida del generador de envolvente Osc. 3.
D500–D7FF	54528–55295		IMAGENES DEL SID
D800–D8FF	55296–56319		RAM de color (medios bytes).
DC00-DCFF	56320-56575		Adaptador complejo de interface MOS 6526 (CIA #1).
DC00	56320		Port de datos A (Teclado, Joystick, Paddles Lápiz óptico).
		7–0	Escribe columnas del teclado para su rastreo.
		7–6	Lee Paddles en Port A/B (01=A), (10=B).

HEX	DECIMAL	BITS	DESCRIPCION
		4	Botón de fuego Joys. A. 1=Fuego.
		3–2	Botones de fuego Paddles.
		3–0	Dirección Joystick A. (0–15).
DC01	56321		Port de datos B (Teclado, Joystick, Paddles Port de juego 1.
		7–0	Lee valores de las filas del teclado para su rastreo.
		7	Timer B. Salida Togle.
		6	Timer A. Salida Togle.
		4	Botón de fuego Joystick 1. 1=fuego.
		3–2	Botones fuego Paddles.
		3–0	Dirección Joystick 1.
DC02	56322		Registro de dirección de datos PORT A (56320).
DC03	56323		Registro de dirección de datos PORT B (56321).
DC04	56324		Timer A. Byte bajo.
DC05	56325		Timer A. Byte altó.
DC06	56326		Timer B. Byte bajo.
DC07	56327		Timer B. Byte alto.
DC08	56328		Reloj diario. 1/10 segundos.
DC09	56329		Reloj diario. Segundos.
DC0A	56330		Reloj diario. Minutos.
DC0B	56331		Reloj diario. Horas +bandera. AM/PM (bit 7).
DC0C	56332		Buffer de datos síncrono del Bus Serie.
DC0D	56333		Registro de control de la CIA (Lee IRQ/escribe máscara).
		7	Bandera IRQ. 1=Sí.
		4	Bandera de IRQ (lectura de cass/entrada SRQ de bus serie).
		3	Interrupción port serie.
		2	Interrupción alarma reloj diario.
		1	Interrupción Timer B.
		0	Interrupción Timer A.
DC0E	56334		Registro de control A de CIA.
		7	Frecuencia reloj diario. 1=50 Hz. 0=60 Hz.
		6	Modo del Port Serie 1=Sal/ 0=en.

HEX	DECIMAL	BITS	DESCRIPCION
		5	Contador Timer A. 1=señal CNT 0=reloj 02 del sistema.
		4	Fuerza carga del Timer A. 1=SI.
		3	Modo de marcha Timer A. 1=Paso a paso. 0=continuo.
		2	Modo de salida Timer A a PB6 0=pulso/1=Togle.
		1	Salida del Timer A en PB6. 1=Si/0=No.
		0	Parada/Marcha Timer A.1=Marcha 0=parada.
DCOF	56335		Registro de control B CIA.
		7	Ajusta alarma reloj TOD 1=alarma/0=Reloj.
		6–5	Selección modo Timer B: 00=Cuenta pulsos del reloj 02. 01=cuenta transiciones positivas de CNT. 10=Cuenta pulsos Timer A. 11=Cuenta pulsos Timer A mientras CNT sea positivo.
		4–0	Igual al registro de control A de la CIA pero para el Timer B.
DD00–DDFF	56576–56831		MOS 6256 (CIA #2).
DD00	56576		Port de datos A (Bus Serie RS–232, Control de memoria).
		7	Entrada de datos Bus Serie.
		6	Entrada de pulsos del reloj por el bus serie.
		5	Salida del Bus Serie.
		4	Salida de pulsos del reloj por el Bus Serie.
		3	Salida de la señal ATN por el Bus Serie.
		2	Salida de datos RS–232 (Port usuario).
		1–0	Selección de banco de la memoria del VIC-II (normal = 11).
DD01	56577		Port de datos B (Port usuario, RS–232).

HEX	DECIMAL	BITS	DESCRIPCION
		7	Usuario/RS–232 listo para datos.
		7	Usuario/RS–232 Borrado para envio.
		5	Usuario.
		4	Usuario/RS–232 Detección de transporte.
		3	Usuario/RS–232 indicador de timbre.
		2	Usuario/RS–232 terminal de datos lista.
		1	Usuario/RS–232 Petición de envío.
		0	Usuario/RS–232 datos recibidos.
DD02	56578		Registro de dirección de datos Port A.
DD03	56579		Registro de dirección de datos Port B.
DD04	56580		Timer A. Byte bajo.
DD05	56581		Timer A. Byte alto.
DD06	56582		Timer B. Byte bajo.
DD07	56583		Timer B. Byte alto.
DD08	56584		Reloj diario. 1/10 de segundo.
DD09	56585		Reloj diario. Segundos.
DD0A	56586		Reloj diario. Minutos.
DD0B	56587		Reloj diario. Horas + bandera AM/PM (bit 7).
DD0C	56588		Buffer serie E/S síncrono.
DD0D	56589		Control de interrupción de la CIA. (Leer NMIs/escribir masc.).
		7	Bandera NMI (1=NMI)/ajusta bandera de borrado.
		4	FLAG1 NMI (Usuario/RS–232 datos de entrada recibidos).
		3	Interrupción port serie.
		1	Interrupción Timer B.
		0	Interrupción Timer A.
DD0E	56590		Registro de control CIA (A).
		7	Frecuencia reloj diario. 1=50 Hz. 0=60 Hz.
		6	Modo port serie. 1=salida. 0=entrada.

HEX	DECIMAL	BITS	DESCRIPCION
		5	Contador Timer A. 1=Señales CNT, 0=reloj 02 del sistema.
		4	Fuerza carga Timer A. 1=Sí.
		3	Modo del timer A. 1=uno a uno 0=continuo.
		2	Salida de Timer A por PB6. 0=Pulso 1=Togle.
		1	Salida de Timer A por PB6. 1=Sí. 0=No.
		0	Paro/Marcha Timer A. 1=Marcha 0=Paro.
DD0F	56591		Registro de control CIA (B).
		7	Ajuste alarma/TOD. 1=alarma/0=reloj.
		6–5	Selección de modo Timer B: 00=Cuenta pulsos del reloj 02. 01=Cuenta transiciones positivas de CNT.- 10=Cuenta pulsos de Timer A. 11=Cuenta pulsos de Timer A mientras CNT sea positivo.
		4–0	Igual al registro de control A, pero para el Timer B.
DE00–DEFF	56832–57087		Reservado para expansiones futuras de E/S.
DF00–DFFF	57088–57343		Reservado para expansiones futuras de E/S.

CAPITULO 6

GUIA DE ENTRADAS/SALIDAS

INTRODUCCION

Los ordenadores tienen tres características fundamentales: pueden hacer cálculos, tomar decisiones y comunicarse. Los cálculos son fáciles de programar. La mayoría de reglas matemáticas le son conocidas. Programar decisiones no es excesivamente complicado, ya que sólo se deben conocer unas sencillas reglas de lógica. Las comunicaciones son la parte más compleja, ya que sólo disponen de muy pocas reglas sin excepción. Esto no es un descuido en el diseño del ordenador, sino que las reglas están pensadas para poder comunicar virtualmente cualquier cosa, y de varias maneras. La única regla general es la siguiente: Cualquier envio de información debe presentarla de forma que el receptor pueda comprenderla.

SALIDA AL TELEVISOR

La forma más fácil de comunicar en BASIC es la instrucción PRINT. PRINT usa la pantalla de T.V. como periférico de salida, y sus ojos son el periférico de entrada, ya que comprenden la información presentada en pantalla.
Al imprimir en la pantalla, su principal objetivo es dar un formato a la impresión de modo que se pueda leer fácilmente. Usted debe ir probando formatos como un artista: puede usar colores, mayúsculas y minúsculas, símbolos gráficos, etc. Recuerde: por perfecto que sea el programa no estará completo si no muestra de una manera clara y legible los resultados.
La instrucción PRINT usa ciertos códigos de carácter como "comandos" del cursor. La tecla **CRSR** no imprime nada, únicamente mueve el cursor en una dirección determinada. Otros comandos cambian el color, borran la pantalla, insertan y borran espacios... La tecla **RETURN** tiene el código de carácter (CHR$) 13. Una tabla completa de estos códigos se muestra en el apéndice C.
El lenguaje BASIC dispone de dos funciones que trabajan junto a la instrucción PRINT. TAB posiciona el cursor en la posición deseada desde la izquierda de la pantalla, SPC mueve el cursor a la derecha el número de espacios deseado desde la posición actual.
Los signos de puntuación en la instrucción PRINT sirven para separar y formatear la información. El punto y coma (;) separa dos items sin espacios entre ellos. Si se encuentra un PRINT seguido de punto y coma, el próximo PRINT se imprimirá a continuación del anterior, en lugar de regresar al principio de la línea siguiente. El punto y coma suprime el retorno de carro ejecutado normalmente al finalizar una instrucción PRINT.
La coma (,) separa los items en columnas. El Commodore 64 tiene 4 columnas de 10 caracteres cada una en la pantalla. Cuando el ordenador imprime una coma el cursor se mueve a la derecha hasta el inicio de la próxima columna. Si la posición actual del cursor sobrepasa el inicio de la última columna la información se imprimirá al inicio de la línea siguiente. Al igual que con el punto y coma, si este es el último carácter a imprimir en una línea se suprime el retorno del carro.
Las comillas ("") separan texto literal de variables. La primera comilla de la línea indica el inicio del área literal, y la comilla de cierre finaliza esta área. Sin embargo, en algunos casos se puede omitir la comilla de cierre.
El RETORNO DE CARRO (CHR$ 13) obliga al cursor a desplazarse al inicio de la

próxima línea lógica en la pantalla. La nueva línea lógica no siempre coincide con la nueva línea real. Si usted escribe algo hasta el final de una línea, ésta se enlaza automáticamente con la siguiente. El ordenador no reconoce dos líneas distintas, sino que cree estar en presencia de una línea larga. Los enlaces se encuentran en la tabla de enlace de líneas (line link table) (Vea el mapa de memoria para ver cómo se activa).
Una línea lógica puede tener 1 o 2 líneas de largo, dependiendo de donde se ha escrito. La línea lógica donde se encuentra el cursor determina la línea a donde irá el mismo al pulsarse un **RETURN**. La línea lógica del inicio de la pantalla determina si la pantalla se deslizará (scroll) 1 o 2 líneas a la vez.
Hay otras formas de utilizar la pantalla como periferico de salida. El capítulo de gráficos describe los comandos para crear objetos que se muevan a través de la pantalla. La sección destinada al VIC-II indica cómo cambiar el tamaño y color de la pantalla, y el capítulo de sonido explica el modo de lograr que surjan del altavoz de su televisor melodías y efectos sonoros.

SALIDA A OTROS PERIFERICOS

Con frecuencia es necesario enviar la salida a otros periféricos en lugar de la pantalla como el cassette, la impresora, la unidad de discos o el modem. La instrucción OPEN en BASIC crea un "canal" para comunicar con uno de estos periféricos. Cuando el canal es abierto (OPEN), la instrucción PRINT# envía caracteres a este periférico.

EJEMPLO de las instrucciones OPEN y PRINT#

```
    100 OPEN 4,4:PRINT#4,"ESCRIBIENDO EN IMPRESORA"
    110 OPEN 3,8,3,"0:FICHERO-DISCO,S,W":PRINT#4, "ENVIO A LA
        UNIDAD DE DISCO"
120 OPEN 1,1,1,"FICHERO-CASS.":PRINT#1, "ENVIO A CASSETTE"
130 OPEN 2,2,0,CHR$(10):PRINT#2, "ENVIO A MODEM"
```

La instrucción OPEN difiere según el periférico a que se destina. Los parámetros de la instrucción OPEN se muestran para cada periférico en la siguiente tabla.

TABLA DE PARAMETROS DE LA INSTRUCCION OPEN

FORMATO: OPEN fichero#,periférico#,número,cadena

PERIFERICO	Num. Perif.	NUMERO	CADENA
CASSETTE	1	0=Entrada 1=Salida 2=Salida con EOT	Nombre fichero
MODEM	2	0	Registros de control
PANTALLA	3	0,1	
IMPRESORA	4,5	0=Mayús.Gráf. 7=Mayús.Minús.	Texto a imprimir
DISCO	de 8 a 11	2–14=Canal de datos. 15=Canal de órdenes.	Unidad#, nombre de fichero, tipo de fichero Comando Read/Write

SALIDA A LA IMPRESORA

La impresora es un periférico de salida similar a la pantalla. El principal problema cuando se envían datos a la impresora es lograr una buena presentación de los mismos. Sus herramientas para lograr esto son: Caracteres invertidos, de doble tamaño, mayúsculas y minúsculas, así como gráficos por puntos programables.
La función SPC trabaja en la impresora del mismo modo que en la pantalla. Sin embargo, la instrucción TAB no trabaja correctamente, ya que tiene en cuenta la posición del cursor en la pantalla, y no la posición de la cabeza de escritura de la impresora.
La instrucción OPEN para la impresora crea un canal para la comunicación. Además especifica el juego de caracteres a emplear, seleccionando entre mayúsculas y gráficos o mayúsculas y minúsculas.

EJEMPLOS de la instrucción OPEN para la impresora:

```
OPEN 1,4:REM MAYUSCULAS Y GRAFICOS
OPEN 1,4,7:REM MAYUSCULAS Y MINUSCULAS
```

Cuando se trabaja con un juego de caracteres, líneas individuales pueden ser impresas en el otro juego. Por ejemplo, si la impresora está en modo mayúsculas y gráficos, se puede pasar al modo de mayúsculas/minusculas mediante la impresión del código de cursor abajo (CHR$(17)). Para volver al modo mayúsculas/minúsculas se emplea el código de cursor arriba (CHR$(145)).
Otras funciones especiales de la impresora se controlan mediante códigos de caracteres. Todos estos códigos simplemente se imprimen, al igual que cualquier otro carácter.

TABLA DE CODIGOS DE CONTROL DE LA IMPRESORA

CODIGO CHR#	PROPOSITO
10	Alimentar línea.
13	RETURN (alimentación de línea automática en las impresoras CBM).
14	Inicio de caracteres de doble ancho.
15	Fin de caracteres de doble ancho.
18	Inico de los caracteres invertidos.
146	Fin de los caracteres invertidos.
17	Cambio al juego de caracteres mayúsc./minúsc.
145	Cambio al juego de caracteres mayúsc/gráficos.
16	Tab a la posición indicada por los siguientes dos caracteres.
27	Movimiento a un punto especifico.
8	Inicio del modo gráfico de puntos programables.
26	Repetición de datos gráficos.

Vea el manual de su impresora Commodore para los detalles de uso de los distintos códigos de control.

SALIDA AL MODEM

El modem es un sencillo periférico que se encarga de traducir los códigos de carácter en impulsos de audio y viceversa, de forma que varios ordenadores se puedan enlazar por medio de las líneas telefónicas. La instrucción OPEN para el modem incluye los parámetros que fijan la velocidad de transmisión y el formato del ordenador con el que desea comunicarse. En la cadena que sigue a la instrucción OPEN deben enviarse dos caracteres para determinar esto.
Las posiciones de los bits del primer código de carácter determinan la velocidad (baudios), el número de bits de los datos, y el número de bits de stop. El segundo código es opcional, y especifica la paridad y duplex de la transmisión. Vea la sección sobre el RS–232 o el manual de su VICMODEM para obtener detalles de este periférico.

EJEMPLO DE LA INSTRUCCION OPEN PARA EL MODEM

```
OPEN 1,2,0,CHR$(6):REM 300 BAUDIOS
100 OPEN 2,2,0,CHR$(163)CHR$(112):REM 110 BAUDIOS, ETC.
```

Muchos ordenadores usan el Código Americano Standard para Intercambio de Información, conocido como ASCII. Este standard difiere de algunos códigos de carácter del código utilizado en el Commodore 64. Si se debe comunicar con otros ordenadores, los códigos de caracteres Commodore deben ser traducidos a sus

equivalentes en ASCII. Una tabla de los códigos ASCII se incluye en el apéndice C de este libro.
La salida al modem es una tarea poco complicada, aparte de la necesidad de convertir los códigos de carácter. Sin embargo, usted debe conocer bien el sistema receptor de la comunicación enviada, especialmente cuando escribe programas que "hablen" con otro ordenador sin intervención humana. Un ejemplo de esto es un programa que envíe automáticamente a un banco de datos su número de cuenta y su clave secreta. Para lograr que esto sea perfecto debe calcular cuidadosamente el número de caracteres a enviar y el número de caracteres de retorno del carro (RETURN). De otra forma, el equipo receptor de la información puede que no la comprenda.

TRABAJANDO CON EL CASSETTE

Los aparatos a cassette disponen de una capacidad casi ilimitada para almacenar datos. Cuando más larga sea la cinta, más información se podrá almacenar en ella. Sin embargo, la limitación de estos aparatos reside en el tiempo de acceso y la velocidad de transmisión. Cuantos mas datos se encuentren en una cinta, más tiempo necesitará el ordenador para encontrar la información deseada.
El programador debe intentar minimizar el factor tiempo cuando trabaja con el cassette. Una práctica común consiste en leer el fichero en cassette entero y trasladarlo a la memoria RAM, realizar todos los procesos sobre esta memoria y reescribir el fichero entero en cassette. Esta técnica le permite examinar, clasificar y modificar sus datos. Sin embargo, el límite para sus ficheros será el de la memoria RAM disponible, lo que no siempre será suficiente.
Si su fichero entero ocupa más memoria RAM que la disponible en su Commodore 64, es probablemente el momento de adquirir una unidad de disco. La unidad de disco puede leer datos de cualquier parte del disco, sin necesidad de leer los datos anteriores. Usted puede escribir las modificaciones precisas encima de los viejos datos sin perjudicar al resto del fichero. Es por esto que la unidad de disco se usa en general para todas las aplicaciones serias (contabilidad, stock, proceso de datos...).
La instrucción PRINT# formatea los datos igual que lo hace la instrucción PRINT. Los símbolos de puntuación trabajan igual. Pero recuerde, usted no está trabajando con la pantalla. El formato se debe efectuar pensando en que los datos serán leídos mediante la instrucción INPUT# o GET#.
Considere la orden PRINT#1,A$,B$,C$. Usándola con la pantalla, las comas entre las variables producen espacios en blanco entre los items para formatear éstos en columnas de diez caracteres de ancho. En el cassette, se añade un número de espacios entre 1 y 10, dependiendo de la longitud de la cadena. Esto reduce el espacio de almacenamiento de su cassette.
Cuando intente leer los datos mediante la orden INPUT#1,A$,B$,C$ el ordenador no encuentra datos para B$ y C$. A$ contiene los datos de las tres variables, más los espacios entre ellas. ¿Qué ha sucedido? Echemos un vistazo al fichero en cassette:

```
A$="PERRO" B$="GATO" C$="VACA"
PRINT#1,A$,B$,C$
```



```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
P E R R O                 G A T O         V A C A RETURN
```

La instrucción INPUT# trabaja como la instrucción INPUT normal. Al escribir datos en contestación a una instrucción INPUT, los diferentes datos se separan pulsando la tecla RETURN o usando comas entre ellos. La instrucción PRINT# coloca un RETURN al final de la línea al igual que la instrucción PRINT normal. A$ contiene las tres cadenas porque no se han separado entre ellas en el cassette, sólo después de grabadas las tres cadenas se ha colocado un RETURN.
Un separador adecuado en el cassette es una coma (,) o un RETURN. El código RETURN se coloca automáticamente al final de una instrucción PRINT o PRINT#. Una forma de colocar un RETURN entre distintos datos es usar una instrucción PRINT# para cada dato a grabar. Un método mejor es asignar a una variable el código CHR$ de RETURN, que es CHR$(13), o una coma. El modo de hacer esto es R$=",":PRINT#1,A$ R$ B$ R$ C$. No use comas u otro signo de puntuación entre los nombres de variable, ya que sólo ocupan memoria en el ordenador y espacio en el cassette.

El fichero escrito de este modo aparecerá de la siguiente forma:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
P E R R O , G A T O , V A C A RETURN
```

La instrucción GET# toma los datos del cassette un solo carácter a la vez. Esta instrucción recibe cada carácter, incluso el código de RETURN y otros signos de puntuación. El código CHR$(0) se interpreta como una cadena vacía, no como una cadena de un carácter con el código 0. Si intenta realizar la función ASC en una cadena vacía recibirá el mensaje de error **ILLEGAL QUANTITY ERROR.**
La línea GET#1, A$: A=ASC(A$) se usa con frecuencia en programas para examinar los datos en cassette. Para evitar mensajes de error, la línea debe modificarse y escribirla así: GET#1, A$:A=ASC(A$+chr$(0)). El CHR$(0) al final es un seguro contra cadenas vacías, sin afectar la función ASC cuando hay otros caracteres en A$.

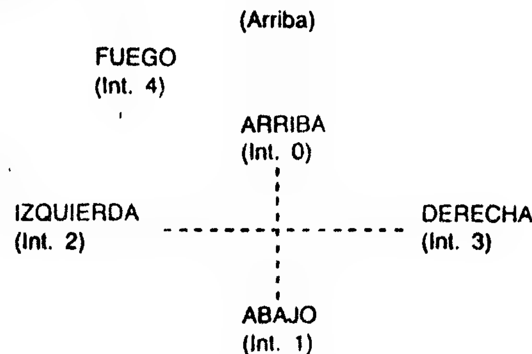
ALMACENAMIENTO DE INFORMACION EN DISCO

Los diskettes tienen 3 formas distintas de almacenamiento. Los ficheros secuenciales son similares a los de cassette, pero pueden utilizarse varios al mismo tiempo. Los ficheros relativos organizan los datos en registros, y entonces se pueden leer y modificar registros individuales en el fichero. Los ficheros de acceso al azar le permiten trabajar con datos en cualquier parte del disco. Están organizados en secciones de 256 bytes llamadas bloques.
Las limitaciones de la instrucción PRINT# se han explicado en la sección del cassette. Las mismas limitaciones se aplican al disco. Son necesarios RETURNs o comas para separar datos. El CHR$(0) es igualmente leído por la instrucción GET# como una cadena vacía.
Los ficheros relativos y de acceso al azar usan "canales" separados de datos y de

comando. Los datos escritos en el disco se envían por el canal de datos, almacenándose en un "buffer" (memoria temporal RAM) del disco. Cuando se ha llenado un registro o bloque, se envía una orden por el canal de comando que le indica a la unidad en que parte del disco debe almacenar la información. Entonces el contenido completo del buffer es escrito en la parte deseada del disco.
Las aplicaciones que requieran grandes cantidades de datos a procesar se almacenan mejor en ficheros relativos de disco. Estos ficheros proporcionan un tiempo de espera mínimo y una gran comodidad para el programador. El manual de la unidad de disco proporciona completa información sobre el uso y forma de programar ficheros en disco.

LOS PORTS DE JUEGO

El Commodore 64 posee dos Ports de Juego de 9 contactos cada uno. Estos ports le permiten conectar mandos de palanca (joystick), raquetas (paddles) o un lápiz óptico. Cada port acepta un joystick o un paddle. El lápiz óptico debe ser conectado en el Port A (únicamente) para control gráfico especial, etc. Esta sección contiene ejemplos de cómo usar los joysticks y paddles desde el BASIC o el lenguaje máquina.
El joystick digital se conecta a la CIA#1 (Adaptador Complejo de Interface MOS 6526). Este periférico de Entrada/Salida también controla los botones de fuego de los paddles y la exploración del teclado. El chip CIA 6526 tiene 16 registros que se encuentran en las posiciones de memoria de 56320 a 56335 inclusive ($DC00 a $DC0F). El Port A se encuentra en la posición 56320 ($DC00) y el Port B está en la posición 56321 ($DC01).
Un joystick digital tiene cinco interruptores distintos, cuatro de los cuales se usan para determinar la dirección y uno para el botón de fuego. El joystick está dispuesto de la siguiente forma:



Estos interruptores corresponden a los 5 bits menos significativos de los registros 56320 o 56321. Normalmente los bits están a 1 si NO se escoge dirección o el botón de fuego NO está pulsado. Cuando el botón de fuego está pulsado, el bit correspondiente (bit 4 en este caso) cambia a 0. Para leer el joystick desde BASIC se puede usar la siguiente subrutina:


```
10 FORK=0TO10:REM PONE LA DIRECCION DEL STRING
20 READDR$(K):NEXT
30 DATA"","N","S","","W","NW"
40 DATA"SW","","E","NE","SE"
50 PRINT"GOING...";
60 GOSUB100:REM LEE EL JOYSTICK
65 IFDR$(JV)=""THEN60:REM MIRAR SI HA SIDO ESCOGIDA UNA DIRECCION
70 PRINTDR$(JV);"";:REM DICE QUE DIRECCION
80 IFFR=16THEN60:REM MIRA SI HA SIDO PULSADO EL BOTON DE DISPARO
90 PRINT"-----F-----I-----R-----E-----!!!":GOTO60
100 JV=PEEK(56320):REMCOGE EL VALOR DEL JOYSTICK
110 FR=JVAND16:REM FORMA EL STATUS DEL BOTON DE DISPARO
120 JV=15-(JVAND15):REM FORMA EL VALOR DE LA DIRECCION
130 RETURN
```

NOTA: Para el segundo joystick, utilice JV=PEEK(56321)

Los valores de JV corresponden a las siguientes direcciones:

JV IGUAL A	DIRECCION
0	NINGUNA
1	ARRIBA
2	ABAJO
3	–
4	IZQUIERDA
5	ARRIBA E IZQUIERDA
6	ABAJO E IZQUIERDA
7	–
8	DERECHA
9	ARRIBA Y DERECHA
10	ABAJO Y DERECHA

Una pequeña rutina en código máquina que cumpla la misma tarea puede ser la siguiente:

```
1000 .PAGE 'JOYSTICK.8/5'  JOYSTICK - BUTTON READ
ROUTINE
1010 ;
1020 ;AUTHOR - BILL HINDORFF
1030 ;
1040 DX=$C110
1050 DY=$C111
1060 *=$C200
1070 DJRR   LDA $DC00      ;     (GET INPUT FROM PORT
A ONLY)
1080 DJRRB  LDY #0         ;THIS ROUTINE READS AND
```

```
DECODES THE
1090          LDX #0          ;JOYSTICK/FIREBUTTON
INPUT DATA IN
1100          LSR A           ;THE ACCUMULATOR. THIS
LEAST SIGNIFICANT
1110          BCS DJR0        ;5 BITS CONTAIN THE
SWITCH CLOSURE
1120          DEY             ;INFORMATION. IF A SWITCH
IS CLOSED THEN IT
1130 DJR0     LSR A           ;PRODUCES A ZERO BIT. IF
A SWITCH IS OPEN THEN
1140          BCS DJR1        ;IT PRODUCES A ONE BIT.
THE JOYSTICK DIR-
1150          INY             ;ECTIONS ARE RIGHT, LEFT,
FORWARD, BACKWARD
1160 DJR1     LSR A           ;BIT3=RIGHT, BIT2=LEFT,
BIT1=BACKWARD,
1170          BCS DJR2        ;BIT0=FORWARD AND
BIT4=FIRE BUTTON.
1180          DEX             ;AT RTS TIME DX AND DY
CONTAIN 2'S COMPLIMENT
1190 DJR2     LSR A           ;DIRECTION NUMBERS I.E.
$FF=-1, $00=0, $01=1.
1200          BCS DJR3        ;DX=1 (MOVE RIGHT), DX=-1
(MOVE LEFT),
1210          INX             ;DX=0 (NO X CHANGE).
DY=-1 (MOVE UP SCREEN),
1220 DJR3     LSR A           ;DY=1 (MOVE DOWN SCREEN),
DY=0 (NO Y CHANGE).
1230          STX DX          ;THE FORWARD JOYSTICK
POSITION CORRESPONDS
1240          STY DY          ;TO MOVE UP THE SCREEN
AND THE BACKWARD
1250          RTS             ;POSITION TO MOVE DOWN
SCREEN.
1260 ;
1270 ;AT RTS TIME THE CARRY FLAG CONTAINS THE FIRE
BUTTON STATE.
1280 ;IF C=1 THEN BUTTON NOT PRESSED. IF C=0 THEN
PRESSED.
1290 ;
1300 .END
```

PADDLES (Raquetas)

Un paddle se conecta a la CIA#1 y al chip SID (Periférico de Interface de Sonido MOS 6581) a través del port de juego. El valor del paddle es leido por los registros del chip SID 54297 ($D419) y 54298 ($D41A). LOS PADDLES NO SE PUEDEN LEER CON SEGURIDAD UTILIZANDO SOLO EL BASIC!!!! La mejor forma de usar los paddles, desde BASIC o desde código máquina, es usar la siguiente rutina en lenguaje máquina...(SYS a ella desde BASIC, entonces PEEK las posiciones de memoria utilizadas por la rutina).


```
1000
;**************************************************************
1010 ;* FOUR PADDLE READ ROUTINE (CAN ALSO BE USED
FOR TWO)
1020
;**************************************************************
1030 ;AUTHOR - BILL HINDORFF
1040 PORTA=$DC00
1050 CIDDRA=$DC02
1060 SID=$D400
1070 *=$C100
1080 BUFFER *=*+1
1090 PDLX *=*+2
1100 PDLY *=*+2
1110 BTNA *=*+1
1120 BTNB *=*+1
1130 *=$C000
1140 PDLRD
1150      LDX #1                  ;FOR FOUR PADDLES
OR TWO ANALOG JOYSTICKS
1160 PDLRD0                       ;ENTRY POINT FOR
ONE PAIR (CONDITION X 1ST)
1170      SEI
1180      LDA CIDDRA              ;GET CURRENT VALUE
OF DDR
1190      STA BUFFER              ;SAVE IT AWAY
1200      LDA #$C0
1210      STA CIDDRA              ;SET PORT A FOR
INPUT
1220      LDA #$80
1230 PDLRD1
1240      STA PORTA               ;ADDRESS A PAIR OF
PADDLES
1250      LDY #$80                ;WAIT A WHILE
1260 PDLRD2
1270      NOP
1280      DEY
1290      BPL PDLRD2
1300      LDA SID+25              ;GET X VALUE
1310      STA PDLX,X
1320      LDA SID+26              ;GET Y VALUE
1330      STA PDLY,X
1340      LDA PORTA               ;TIME TO READ
PADDLE FIRE BUTTONS
1350      ORA #$80                ;MAKE IT THE SAME
AS OTHER PAIR
1360      STA BTNA                ;BIT 2 IS PDL X,
BIT 3 IS PDL Y
1370      LDA #$40
1380      DEX                     ;ALL PAIRS DONE?
1390      BPL PDLRD1              ;NO
1400      LDA BUFFER
1410      STA CIDDRA              ;RESTORE PREVIOUS
VALUE OF DDR
1420      LDA PORTA+1             ;FOR 2ND PAIR -
```

```
1430    STA BTNB                    ;BIT 2 IS PDL X,
BIT 3 IS PDL Y
1440    CLI
1450    RTS
1460 .END
```

Los paddles pueden ser leídos usando el siguiente programa en BASIC:

```
10 C=12*4096:REM SET PADDLE ROUTINE START
11 REM POKE IN THE PADDLE READING ROUTINE
15 FORI=0TO63:READA:POKEC+I,A:NEXT
20 SYSC:REM CALL THE PADDLE ROUTINE
30 P1=PEEK(C+257):REM SET PADDLE ONE VALUE
40 P2=PEEK(C+258):REM  "    "    TWO   "
50 P3=PEEK(C+259):REM  "    "   THREE  "
60 P4=PEEK(C+260):REM  "    "    FOUR  "
61 REM READ FIRE BUTTON STATUS
62 S1=PEEK(C+261):S2=PEEK(C+262)
70 PRINTP1,P2,P3,P4:REM PRINT PADDLE VALUES
72 REM PRINT FIRE BUTTON STATUS
75 PRINT:PRINT"FIRE A ";S1,"FIRE B ";S2
80 FORW=1TO50:NEXT:REM WAIT A WHILE
         SHIFT CLR/HOME
90 PRINT"♥":PRINT:GOTO 20:REM CLEAR SCREEN AND DO
AGAIN
95 REM DATA FOR MACHINE CODE ROUTINE
100 DATA162,1,120,173,2,220,141,0,193,169,192,141,
2,220,169
110 DATA128,141,0,220,160,128,234,136,16,252,173,
25,212,157
120 DATA1,193,173,26,212,157,3,193,173,0,220,9,128,
141,5,193
130 DATA169,64,202,16,222,173,0,193,141,2,220,173,
1,220,141
140 DATA6,193,88,96
```

LAPIZ OPTICO

El lápiz óptico coloca el valor de la posición de la pantalla que está explorando en un par de registros (LPX,LPY) con un estrecho margen de error. El registro de posición X 19 ($13) contiene los 8 MSB de la posición X en el momento de la exploración. Puesto que la posición de X está definida por un contador de 512 posiciones (9 bits), se obtiene una resolución de dos puntos horizontales. De forma similar, la posición de Y se almacena en el registro 20 ($14), pero al tener soló 8 bits se obtiene la posición de barrido dentro de la pantalla. La lectura del lápiz óptico se efectúa una vez por cuadro, y las subsiguientes lecturas en el mismo cuadro no tienen efecto. Por esto, usted deberá tomar varias muestras antes de colocar el lápiz hacia la pantalla (3 o más muestras, dependiendo de las características de su lápiz óptico).

DESCRIPCION DEL INTERFACE RS-232

DESCRIPCION GENERAL

El Commodore 64 dispone de un interface RS-232 incorporado para la conexión de cualquier modem RS-232, impresora u otro periférico. Para conectar un periférico al Commodore 64 todo lo que necesita es un cable y un poco de programación.
El RS-232 del Commodore 64 está dispuesto en el formato standard de este interface, pero los voltajes tienen niveles TTL (de 0 a 5V) en lugar de los niveles normales del RS-232 (de −12 a 12V). El cable entre el Commodore 64 y el periférico debe tener en cuenta esta necesaria conversión de voltaje. El cartucho de Interface Commodore RS-232 cumple este propósito.
El software para ajustar el interface puede ser construido a partir de BASIC o mediante el KERNAL para los programas en código máquina.
El software de interface RS-232 en BASIC utiliza las instrucciones normales en BASIC: OPEN, CLOSE, CMD, INPUT#, GET#, PRINT# y la variable reservada ST. INPUT# y GET# toman los datos desde el buffer receptor, mientras que PRINT# y CMD colocan los datos en el buffer emisor. El uso de estas instrucciones así como ejemplos serán mostrados más adelante en este capítulo.
El KERNAL del RS-232 a nivel de byte y de bit funciona bajo el control de la CIA#2 6526, temporizadores e interrupciones. El chip 6526 genera NMI (Interrupciones no enmascarables) requeridas para el proceso del RS-232. Esto permite que el proceso del RS-232 pueda tener lugar simultáneamente con la ejecución de programas en BASIC o lenguaje máquina. El KERNAL dispone también de rutinas para proteger de la disrrupción causada por las NMI del RS-232 a los datos a enviar al cassette o diskette. Durante las actividades del cassette o diskette NO pueden ser recibidos datos desde un periférico RS-232. Pero gracias a que estos apartados son sólo locales (asumiendo que ha elaborado los programas cuidadosamente) no se produce ninguna interferencia en los resultados.
Hay dos buffers en el interface RS-232 de su Commodore 64 para prevenir la pérdida de datos durante la emisión y recepción de los mismos.
Los buffers del KERNAL de su Commodore 64 consisten en dos buffers FIFO (First–In/First–Out), cada uno de 256 bytes de capacidad, situados en la zona más alta de la memoria. La apertura (OPEN) de un canal RS-232 automáticamente reserva 512 bytes de memoria para estos buffers. Si no hay suficiente espacio de memoria al final del programa no aparecerá mensaje de error, pero su programa puede ser destruido. ¡VAYA CON CUIDADO!
Estos buffers son cancelados automáticamente al ejecutar la instrucción CLOSE.

APERTURA DE UN CANAL RS-232

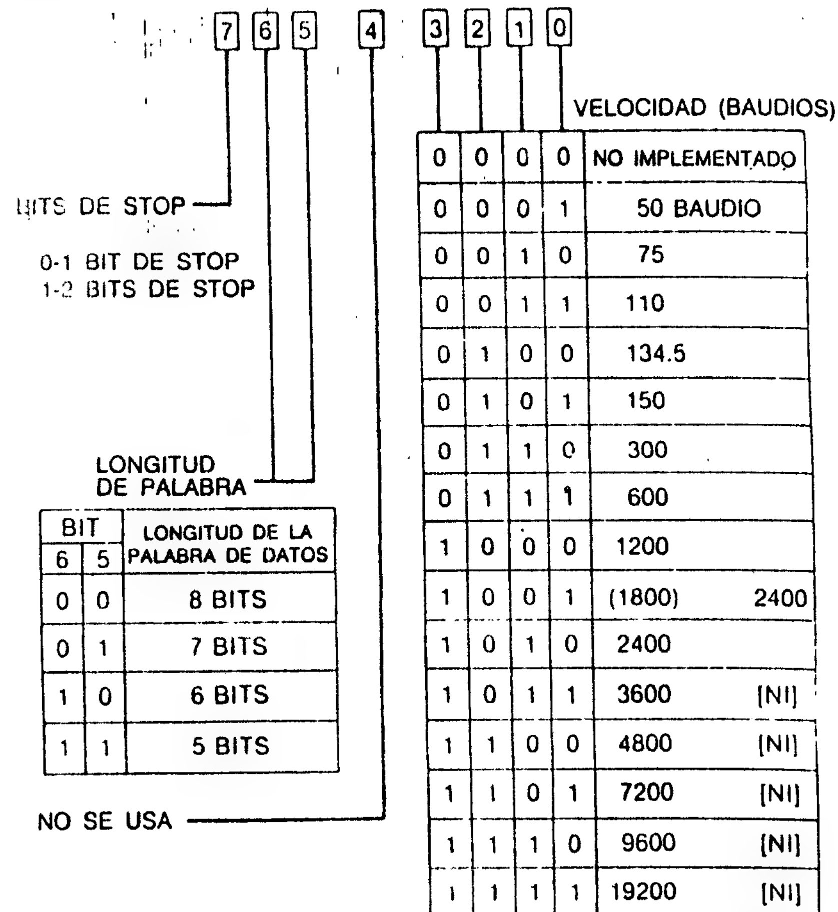
Sólo se puede abrir un canal RS-232 a la vez; una segunda instrucción OPEN causa el reajuste de los punteros de buffer. Todos los caracteres en cualquiera de los dos buffers se perderán.
Se pueden colocar hasta 4 caracteres en el campo del nombre de fichero. Los dos primeros son los caracteres de control y comando; los otros dos están reservados

para sistemas futuros. La velocidad de transmisión, paridad y otras opciones se pueden seleccionar usando esta característica.
No se chequean los errores en el carácter de control para detectar una velocidad de transmisión no implementada. Cualquier carácter de control ilegal causa que el sistema envíe los datos de salida a una velocidad muy lenta (menos de 50 baudios).

SINTAXIS EN BASIC

OPEN lfn,2,0,"<registro de control><registro de comando><opt baudios bajo><opt baudios alto>"

lfn - El número de fichero lógico que puede ser cualquier número entre 1 y 255. Si escogo un número superior a 127 se alimentará una línea después de cada retorno de carro.



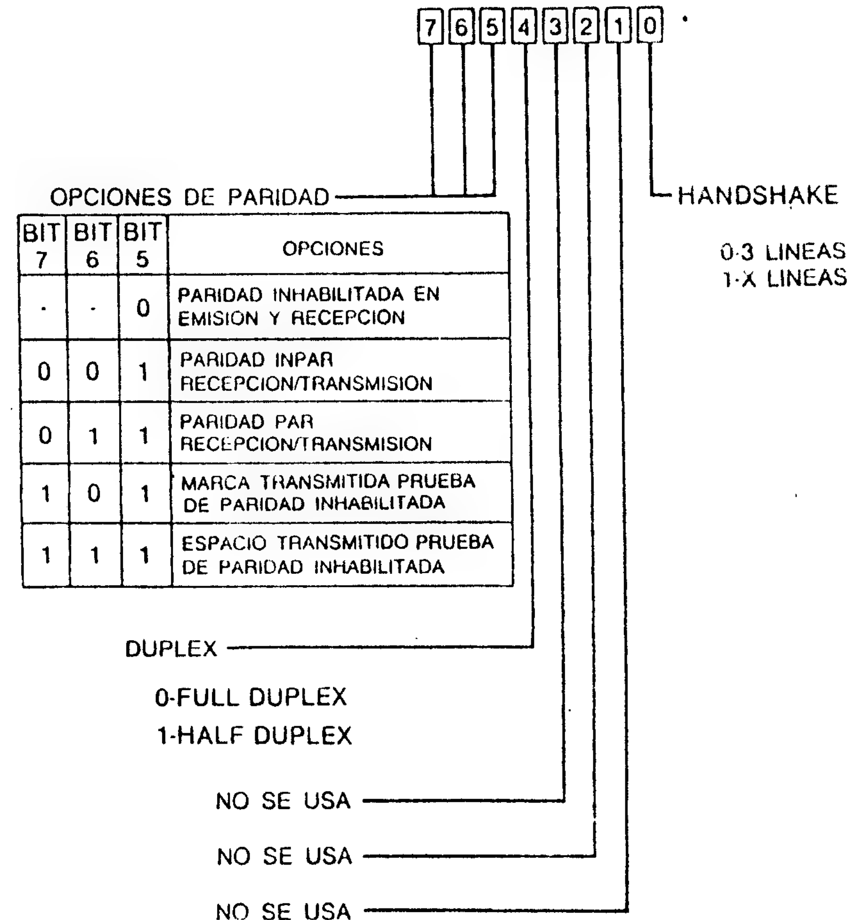
BIT 6	BIT 5	LONGITUD DE LA PALABRA DE DATOS
0	0	8 BITS
0	1	7 BITS
1	0	6 BITS
1	1	5 BITS

3	2	1	0	VELOCIDAD (BAUDIOS)	
0	0	0	0	NO IMPLEMENTADO	
0	0	0	1	50 BAUDIO	
0	0	1	0	75	
0	0	1	1	110	
0	1	0	0	134.5	
0	1	0	1	150	
0	1	1	0	300	
0	1	1	1	600	
1	0	0	0	1200	
1	0	0	1	(1800)	2400
1	0	1	0	2400	
1	0	1	1	3600	[NI]
1	1	0	0	4800	[NI]
1	1	0	1	7200	[NI]
1	1	1	0	9600	[NI]
1	1	1	1	19200	[NI]

Fig 6-1. Mapa de Registros de Control.

<registro de control>- Es un carácter de un solo byte (vea Fig 6-1) requerido para especificar la velocidad de transmisión en baudios. Si los 4 bits más bajos de este carácter estan a cero (0), los caracteres <opt baudios bajo><opt baudios alto> dan la correspondiente velocidad basándose en lo siguiente:

<opt baudios bajo>=<frecuencia del sistema/tasa/2–100–<opt baudios alto>*256
<opt baudios alto>=INT((frecuencia del sistema/tasa/2–100)/256



BIT 7	BIT 6	BIT 5	OPCIONES
-	-	0	PARIDAD INHABILITADA EN EMISION Y RECEPCION
0	0	1	PARIDAD INPAR RECEPCION/TRANSMISION
0	1	1	PARIDAD PAR RECEPCION/TRANSMISION
1	0	1	MARCA TRANSMITIDA PRUEBA DE PARIDAD INHABILITADA
1	1	1	ESPACIO TRANSMITIDO PRUEBA DE PARIDAD INHABILITADA

Fig 6-2. Mapa de Registros de Comando.

Las fórmulas anteriores se basan en los siguientes hechos:

Frecuencia del sistema = 1.02273E6 NTSC (T.V. en Estados Unidos)
= 0.98525E6 PAL (T.V. en la mayoría de los países de Europa, entre ellos España).

<registro de comando>–Es un carácter de un solo byte (Vea Fig. 6-2) que define otros parámetros del terminal. Este carácter NO es imprescindible.

ENTRADA DEL KERNAL

OPEN ($FFC0) (Vea las especificaciones del KERNAL para obtener mayor información).

> **NOTA IMPORTANTE:** En un programa en BASIC, la apertura de un canal RS-232 debe hacerse ANTES de definir cualquier variable o tabla porque un CLR se ejecuta automáticamente cuando se abre un canal RS 232 (Esto sucede porque se deben reservar 512 bytes para los buffers). Recuerde también que su programa puede ser destruido si no se dispone de 512 bytes libres antes de la apertura de un canal RS-232.c

RECEPCION DE DATOS DESDE UN CANAL RS-232

Cuando se reciben datos de un canal RS-232, el buffer receptor del Commodore 64 puede almacenar 255 caracteres antes de llenarse por completo. Esto se indica por la variable de estado del RS-232 (ST en BASIC, o RSSTAT en código máquina). Si ocurre un desbordamiento de la capacidad del buffer, todos los caracteres recibidos desde que el buffer estaba completo se perderán. Obviamente, se debe procurar recoger rápidamente la información residente en el buffer para que éste no llegue nunca a llenarse.

Si desea recibir datos desde un periférico RS-232 a gran velocidad deberá utilizar un programa en lenguaje máquina, ya que el BASIC es demasiado lento y probablemente perdería caracteres.

SINTAXIS EN BASIC

Recomendada: GET#lfn,<variable de cadena>
NO recomendada: INPUT#lfn,<lista de variables>

ENTRADAS DEL KERNAL

CHKIN($FFC6)-Vea la sección sobre el KERNAL para más detalles
GETIN($FFE4)-Vea la sección sobre el KERNAL para más detalles
CHRIN($FFCF)-Vea la sección sobre el KERNAL para más detalles

> **NOTAS:**
> Si la palabra es menor de 8 bits, todos los bits no usados contendrán el valor cero.
> Si un GET# no encuentra datos en el buffer, se devuelve una cadena vacía ("").
> Si se usa INPUT#, el sistema espera hasta que encuentra una cadena no vacía seguida de retorno del carro Por esto, si las señales Clear to Send (CTS) o Dataset Ready (DSR) desaparecen durante la ejecución de INPUT#, el sistema se queda en estado de solo RESTORE. Es por esto por lo que NO se recomienda el uso de INPUT# o CHRIN (rutina equivalente del KERNAL).
> La rutina CHKIN maneja la línea x de acuerdo con el standard de la EIA para los interfaces RS-232 (Agosto de 1979). Las señales Request to send (RTS), CTS y Received line signal (DCD) se han implementado en el Commodore 64 definido como un terminal de datos.

ENVIO DE DATOS A UN CANAL RS-232

Cuando se envían datos, el buffer de salida puede contener 255 caracteres antes de que se desborde. El sistema puede esperar en la rutina CHROUT hasta que se permita la transmisión o se pulsen las teclas RUN/STOP y RESTORE.

SINTAXIS EN BASIC

CMDlfn–actúa igual que en las especificaciones del BASIC
PRINT#lfn,<lista de variables>

ENTRADAS DEL KERNAL

CHKOUT($FFC9)–Vea la sección del KERNAL para más información.
CHROUT($FFD2)–Vea la sección del KERNAL para más información.

> **NOTA IMPORTANTE:** No existe un retardo del retorno del carro en el canal de salida. Esto significa que una impresora normal RS–232 puede no imprimir bien a menos que se implemente algún tipo de "HOLD-OFF" (se pida al Commodore 64 que espere) o exista algún "BUFFER" en la impresora. El "HOLD-HOFF" se puede improvisar fácilmente en el programa. Si se trabaja con la señal CTS (x-líneas) el Commodore 64 realizará un "HOLD OFF" hasta que se permita la transmisión.
> La rutina CHKOUT maneja el handshake de x-líneas que sigue las normas EIA (agosto de 1979) para los interfaces RS–232–C. Las líneas RTS, CTS y DCD se han implementado con el Commodore 64 como dispositivo terminal de datos.

CIERRE DE UN CANAL RS–232

El cierre de un canal RS–232 descarta todos los datos presentes en los buffers en el momento de la ejecución (hayan o no sido transmitidos o impresos), para todas las transmisiones y recepciones del RS-232, ajusta la señal RTS y las líneas altas (S_{out}) de datos transmitidos, y destruye los dos buffers del RS-232.

SINTAXIS EN BASIC

CLOSElfn

ENTRADA DEL KERNAL

CLOSE($FFC3)–Vea la sección correspondiente al KERNAL para más información.

NOTA: Asegúrese de que se han transmitido todos los datos antes de cerrar un canal RS-232. Una forma de chequear esto desde BASIC es:

```
100 SS=ST:IF(SS=0 OR SS=8) THEN100
110 CLOSE lfn
```

Tabla 6-1. Líneas del Port de Usuario

(6526 DEVICE #2 Loc. $DD00 - $DD0F)						
PIN ID	6526 ID	DESCRIPCION	EIA	ABV	IN/OUT	MODOS
C	PB0	RECEIVED DATA	(BB)	S_{in}	IN	1 2
D	PB1	REQUEST TO SEND	(CA)	RTS	OUT	1*2
E	PB2	DATA TERMINAL READY	(CD)	DTR	OUT	1*2
F	PB3	RING INDICATOR	(CE)	RI	IN	3
H	PB4	RECEIVED LINE SIGNAL	(CF)	DCD	IN	2
J	PB5	UNASSIGNED	( )	XXX	IN	3
K	PB6	CLEAR TO SEND	(CB)	CTS	IN	2
L	PB7	DATA SET READY	(CC)	DSR	IN	2
B	FLAG2	RECEIVED DATA	(BB)	S_{in}	IN	1 2
M	PA2	TRANSMITTED DATA	(BA)	S_{out}	OUT	1 2
A	GND	PROTECTIVE GROUND	(AA)	GND		1 2
N	GND	SIGNAL GROUND	(AB)	GND		1 2 3

MODOS:

1) 3-LINE INTERFACE (S_{in},S_{out},GND)

2) X-LINE INTERFACE

3) Utilizable por el usuario (no usado)

* Estas líneas permanecen altas en MODO 3-LINE

```
[7] [6] [5] [4] [3] [2] [1] [0] (Machine Lang. — RSSTAT
 :   :   :   :   :   :   :   :__PARITY ERROR BIT
 :   :   :   :   :   :   :______FRAMING ERROR BIT
 :   :   :   :   :   :__________RECEIVER BUFFER OVERRUN BIT
 :   :   :   :   :______________RECEIVER BUFFER—EMPTY
 :   :   :   :                  (USE TO TEST AFTER A GET#)
 :   :   :   :__________________CTS SIGNAL MISSING BIT
 :   :   :______________________UNUSED BIT
 :   :__________________________DSR SIGNAL MISSING BIT
 :______________________________BREAK DETECTED BIT
```

Figura 6-3. Registro de Estado (ST) del RS–232.

NOTAS:

Si los BITS están a cero no se detecta error.

El registro de estado del RS–232 puede ser leído desde BASIC usando la variable ST. Si ST es leído desde BASIC o usando la rutina del KERNAL READST la palabra de estado del RS-232 es borrada. Si se precisan múltiples usos de la palabra de estado, ésta debe ser asignada a otra variable. Por ejemplo:

```
SR=ST:REM ASIGNA ST A SR
```

El estado del RS 232 puede ser leído (y borrado) sólo cuando el canal RS–232 ha sido el último en utilizarse para una operación de E/S.

PROGRAMAS DE MUESTRA EN BASIC

```
10 REM ESTE PROGRAMA ENVIA Y RECIBE DATOS A/DE UN SILENT 700
11 REM TERMINAL MODIFICADO PARA PET ASCII
20 REM TI SILENT 700 COLOCADO:300 BAUDIOS,ASCII DE 7 BITS,PARIDAD IMPAR
21 REM FULL DUPLEX
30 REM MISMA COLOCACION EN EL ORDENADOR QUE USANDO UN INTERFASE DE 3 LINEAS
100 OPEN 2,2,3,CHR$(6+32)+CHR$(32+128):REM ABRIR EL CANAL
110 GET#2,A$:REM CONECTAR EL CANAL RECEPTOR(TIRO A INVALIDA)
200 REM BUCLE PRINCIPAL
210 GET  B$:REM COGE DEL TECLADO DEL ORDENADOR
220 IF B$<>"" THEN PRINT#2,B$:REM SI SE PULSA UNA TECLA,SE ENVIA AL TERMINAL
230 GET#2,C$:REM COGE UNA TECLA DEL TERMINAL
240 PRINT B$;C$;:REM IMPRIME LAS ENTRADAS DE DATOS EN LA PANTALLA DEL ORDENADOR
250 SR=ST:IF SR=0 OR SR=8 THEN 200:REM REVISA EL ESTATUS,SI ES BUENO ENTONCES
300 REM AVISA DE ERROR
310 PRINT"ERROR: ";
320 IF SR AND 1 THEN PRINT "PARIDAD"
330 IF SR AND 2 THEN PRINT "ESTRUCTURA"
340 IF SR AND 4 THEN PRINT "BUFFER RECEPTOR LLENO"
350 IF SR AND 128 THEN PRINT "BREAK"
360 IF (PEEK(673) AND 1 THEN 360:REM ESPERA A TODOS LOS CARACTERES
370 CLOSE 2:END
100 OPEN 5,2,3,CHR$(6)
110 DIM F%(255),T%(255)
200 FOR J=32 TO 64:T%(J)=J:NEXT
210 T%(13)=13:T%(20)=8:RV=18:CT=0
220 FOR J=65 TO 90:K=J+32:T%(J)=K:NEXT
230 FOR J=91 TO 95:T%(J)=J:NEXT
240 FOR J=193 TO 218:K=J-128:T%(J)=K:NEXT
250 T%(146)=16:T%(133)=16
260 FOR J=0 TO 255
270 K=T%(J)
```

```
280 IF K<>0THEN F%(K)=J:F%(K+128)=J
290 NEXT
300 PRINT " "CHR$(147)
310 GET#5,A$
320 IF A$=""OR ST<>0 THEN 360
330 PRINT" "CHR$(157);CHR$(F%(ASC(A$)));
340 IF F%(ASC(A$))=34 THEN POKE212,0
350 GOTO 310
360 PRINTCHR$(RV)" "CHR$(157);CHR$(146);:GET A$
370 IF A$<>""THENPRINT#5,CHR$(T%(ASC(A$)));
380 CT=CT+1
390 IF CT=8 THENCT=0:RV=164-RV
410 GOTO310
```

POSICIONES DE LOS PUNTEROS DE BUFFERS RECEPTOR/TRANSMISOR

$00F7-RIBUF–Un puntero de dos bytes indicando la posición de base del buffer receptor.

$00F9-ROBUF–Un puntero de dos bytes indicando la posición de base del buffer transmisor.

Las dos posiciones de arriba son ajustadas por la rutina OPEN del KERNAL, cada una apuntando a un buffer de 256 bytes distinto. Se desactivan escribiendo un cero en los bytes más significativos ($00F8 y $00F9), lo que sucede al ejecutar la rutina CLOSE del KERNAL. También pueden ser activados/desactivados por el programador en lenguaje máquina para sus propios propósitos, es decir para crear únicamente los buffers necesarios. Cuando se usa una rutina en lenguaje máquina para posicionar los buffers se debe tener sumo cuidado en colocar el valor correcto en los punteros de límite de memoria (top of memory), especialmente si programas en BASIC deben ejecutarse simultáneamente.

POSICIONES DE PAGINA CERO Y SU USO EN EL SISTEMA DE INTERFACE RS–232

$00A7-INBIT–Almacenamiento temp. entrada bit receptor
$00A8-BITCI–Cuenta del bit recibido.
$00A9-RINONE–Bandera de inicio de check del bit recibido
$00AA-RIDATA–Posición buffer/assembly del byte recibido
$00AB-RIPRTY–Almacenamiento del bit de paridad recibido
$00B4-BITTS–Cuenta del bit transmitido
$00B5-NXTBIT–Próximo bit a transmitir
$00B6-RODATA–Posición buffer/assembly del byte transmitido

Todas estas posiciones de página cero se usan localmente y solo muestran una guía para comprender las rutinas asociadas. Estas posiciones no pueden ser utilizadas desde BASIC o el KERNAL a nivel de programador para lograr que el RS-232 realice alguna tarea. Deben ser usadas las rutinas para el sistema RS-232.

POSICIONES DE FUERA DE LA PAGINA CERO Y SU USO EN EL SISTEMA DE INTERFACE RS-232

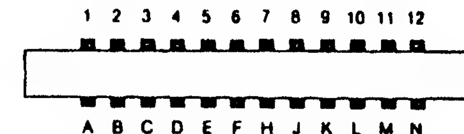
Almacenamiento general del RS-232

$0293–M51CTR–Registro de control del Pseudo 6551 (vea Fig. 6-1)
$0294–M51COR–Registro de comando del PSEUDO 6551 (vea Fig. 6-2)
$0295–M51AJB–Los dos bytes siguientes a los registros de cohtrol y comando en el campo del nombre de fichero. Estas posiciones contienen la velocidad en baudios para el inicio del test de bit durante la actividad del interface.
$0297–RSSTAT–El registro de estado del RS-232 (Vea Figura 6-3)
$0298–BITNUM–El número de bits enviados/recibidos
$0299–BAUDOF–Dos bytes que son iguales al tiempo de un bit (Basado en el sistema de reloj/baudios)

EL PORT DEL USUARIO

El port del usuario es un modo de conectar el Commodore 64 con el mundo exterior Usando las líneas disponibles en este port puede conectar el Commodore 64 a una impresora, un sintetizador de voz, un modem e incluso se puede conectar a otro ordenador.
El port del Commodore 64 está conectado directamente a uno de los chips CIA 6526. Mediante programación, la CIA puede conectar con muchos otros periféricos.

DESCRIPCION DE LOS CONTACTOS DEL PORT (PINS)



DESCRIPCION DE CONTACTOS DEL PORT

CONTACTO CARA SUP.	DESCRIPCION	NOTAS
1	TIERRA	
2	+5V	(100 mA MAX.)
3	RESET	Activando este contacto, el Commodore 64 se reinicia (RESET) completamente. Los punteros del programa en BASIC son reajustados, pero la memoria no se borra. Es también una salida RESET para periféricos externos.
4	CNT1	Contador del port serie desde la CIA#1. (Vea ESPECIF.CIA)
5	SP1	Port serie desde la CIA#1. (Vea ESPECIF. CIA 6526)
6	CNT2	Contador del port serie desde la CIA#2. (Vea ESPECIF.CIA)
7	SP2	Port serie desde la CIA#2. (Vea ESPECIF.CIA 6526)
8	PC2	Línea de handshaking desde la CIA#2 (Vea ESPECIF.CIA)
9	SERIAL ATN	Este contacto está conectado a la línea ATN del bus serie.
10	9 VAC+FASE	Conectado directamente al transformador del Commodore 64 (50 mA MAX.).
11	9 VAC–FASE	
12	GND	
CARA INF.		
A	GND	El Commodore 64 le da control sobre el PORT B del la CIA#1. Ocho líneas de Entradas/Salidas están disponibles, así como dos líneas para handshaking con un periférico exterior. Las líneas de E/S para el PORT B se controlan por dos posiciones. Una es el PORT propiamente dicho, y se encuentra en la posición 56577 ($DD01). Naturalmente, PEEK para leer entradas y POKE para ajustar salidas. Cada una de las 8 líneas de E/S pueden activarse como entrada o salida colocando el valor adecuado en el REGISTRO DE DIRECCION DE DATOS.
B	FLAG2	
C	PB0	
D	PB1	
E	PB2	
F	PB3	
H	PB4	
J	PB5	
K	PB6	
L	PB7	
M	PA2	
N	GND	

El **REGISTRO DE DIRECCION DE DATOS** se encuentra en la posición 56579 ($DD03). Cada una de las ocho líneas del port tiene un BIT en el registro de ocho bits REGISTRO DE DIRECCION DE DATOS (DDR), que controla las líneas que sirven para entradas y las que sirven para salidas. Si un bit del DDR está a 1, la correspondiente línea en el port será de SALIDA. Si el bit en el DDR está a 0, la línea correspondiente en el port será de ENTRADA. Por ejemplo, si el bit 3 del DDR está a 1, la línea 3 del port será de salida. Un ejemplo adicional:
Si el DDR está ajustado del siguiente modo:

BIT # :76543210
VALOR:00111000

Usted puede ver que las líneas 5, 4 y 3 son de salida, puesto que sus bits asociados están a 1. El resto de las líneas, al tener sus bits a 0, serán de entrada.
Para PEEK o POKE el port de usuario, es necesario usar el DDR además del port propiamente dicho.

Recuerde que las instrucciones PEEK y POKE precisan un número entre 0 y 255. Los números dados en el ejemplo de uso del DDR deben ser traducidos a un número decimal antes de poder usarse. El valor resultante es:

$$2^5 + 2^4 + 2^3 = 32 + 16 + 8 = 56$$

Advierta que el bit# para el DDR es el mismo número que 2 elevado a la potencia del lugar de orden del bit.

(16=2 ↑ 4=2X2X2X2, 8=2 ↑ 3=2X2X2)

Las otras dos líneas, FLAG1 y PA2 son distintas del resto del port de usuario. Estas dos líneas se utilizan principalmente para handshaking, y se programan de distinta forma desde el port B.
El handshaking es necesario cuando se comunican dos periféricos. Puesto que uno de los Periféricos puede ir a mayor velocidad que el otro es necesario dar a los periféricos un modo de que sepan que es lo que está haciendo el otro periférico. Incluso cuando los periféricos trabajan a la misma velocidad, el handshaking es necesario para que los periféricos sepan si el otro va a enviar datos o si los ha recibido correctamente. La línea FLAG1 posee unas características especiales que la hacen idónea para este trabajo.
FLAG1 es una entrada sensitiva a un corte negativo que puede ser usada como una entrada de interrupción de propósitos generales. Cualquier transición negativa en la línea FLAG1 activa el bit de interrupción FLAG. Si el bit de interrupción FLAG es activado, se causa una PETICION DE INTERRUPCION. Si el bit FLAG no es activado, puede ser encabezado desde el registro de interrupción bajo control del programa.
PA2 es el bit 2 del port A de la CIA. Es controlado al igual que cualquier otro bit del port. El port está situado en la posición 56576 ($DD00). El registro de dirección de datos se encuentra en la posición 56578 ($DD02).

EL BUS SERIE

El bus serie está dispuesto en forma de una cadena en margarita diseñada para permitir al Commodore 64 comunicarse con periféricos como el VIC-1541 (Unidad de Disco) o el VIC-1525 (Impresora Gráfica). La ventaja del bus serie es que se le puede conectar más de un periférico. Pueden conectarse hasta cinco periféricos al mismo tiempo en este port.

Hay tres tipos de operación sobre el bus serie: CONTROL, EMISION y RECEPCION. El periférico de CONTROL es el encargado de controlar la operación del bus serie. El TRANSMISOR transmite datos por el bus. El RECEPTOR recibe datos desde el bus.

El Commodore 64 es el controlador del bus. También actúa como EMISOR (por ejemplo cuando envía datos a la impresora) y como RECEPTOR (por ejemplo cuando carga un programa desde el disco). Los periféricos pueden ser de entrada (la impresora), de salida o ambas cosas a la vez (la unidad de disco). Sólo el Commodore 64 puede actuar como controlador.

Todos los periféricos conectados al bus serie pueden recibir los datos transmitidos por el bus. Para lograr que el Commodore 64 transmita los datos al periférico adecuado, cada uno de ellos posee una DIRECCION de bus. Usando esta dirección de periférico el Commodore 64 puede controlar los accesos al bus. Las direcciones en el bus serie van de 4 a 31.

El Commodore 64 puede ordenar a un periférico que emita o reciba datos. Si el Commodore 64 ordena a un periferico que ENVIE datos, iniciará este envío hacia el bus. Si el Commodore ordena a un periférico que RECIBA datos, el periférico que recibe la orden quedará listo para recibir los datos que se le envían. (Desde el Commodore 64 u otro periférico conectado al bus). Sólo un periférico a la vez puede ENVIAR datos; de otro modo habría una confusión en el sistema. Sin embargo, cualquier número de periféricos pueden RECIBIR datos al mismo tiempo que uno los EMITE.

DIRECCIONES NORMALES DEL BUS SERIE

NUMERO	PERIFERICO
4 o 5	IMPRESORA GRAFICA 1525
6	PLOTTER VL-1520
8	UNIDAD DE DISCO 1541

Otras direcciones de periférico son también posibles. Cada periférico dispone de su propia dirección. Algunos de ellos, como la impresora gráfica 1525, permiten la elección entre dos números de dirección a la conveniencia del usuario.

La DIRECCION SECUNDARIA permite al ordenador transmitir información sobre la puesta en marcha al periférico adecuado. Por ejemplo, para establecer conexión con la impresora y que ésta se coloque en el modo MAYUSCULAS/MINUSCULAS, use lo siguiente:

```
OPEN 1,4,7
```

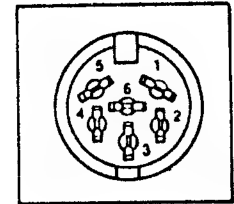

donde:

1 es el número de fichero lógico. (El número que utiliza en la instrucción PRINT#)
4 es la DIRECCION de la impresora, y
7 es la DIRECCION SECUNDARIA que indica a la impresora que entre en el modo MAYUSCULAS/MINUSCULAS.

Se usan seis líneas en el bus serie –3 de entrada y 3 de salida–. Las tres líneas de entrada reciben las señales de datos, control y tiempo, enviándolas al Commodore 64. Las tres líneas de salida envían a los periféricos conectados al bus serie las líneas de datos, control y tiempo.

CONTACTOS DEL BUS SERIE

PIN	DESCRIPTION
1	SRQ IN SERIE
2	TIERRA
3	ATN E/S SERIE
4	CLK E/S SERIE
5	DATOS E/S SERIE
6	SIN CONEXION



SRQ IN SERIE: (PETICION DE SERVICIO SERIE)

Cualquier periférico en el bus serie puede enviar esta señal baja cuando precisa atención del Commodore 64. El Commodore 64 toma entonces el control del periférico.

ATN E/S SERIE:(ATENCION E/S SERIE)

El Commodore 64 usa esta señal para iniciar una secuencia de comandos para un periférico del bus serie. Cuando el Commodore 64 envía esta señal baja, todos los periféricos conectados al bus serie inician la "escucha" para que el Commodore 64 transmita la dirección. El periférico al que se dirige la dirección debe responder en un lapso de tiempo predeterminado; de otro modo, el Commodore 64 asume que dicho periférico no está conectado al bus, y retorna un error en el registro de estado (ST).

CLK E/S SERIE: (RELOJ E/S SERIE)

Esta señal se usa para controlar el tiempo en que se envian los datos en el bus serie.

DATOS E/S SERIE:

Los datos en el bus serie se transmiten bit a bit por esta línea.

EL PORT DE EXPANSION

El conector de expansión es un conector hembra de 44 contactos (22/22) situado en la parte trasera del Commodore 64. Con el Commodore 64 encarado hacia usted, el conector de expansión se encuentra a la derecha del ordenador, en la parte trasera. Para usar este conector se requiere un conector macho de 44 contactos (22/22). Este port se usa para expansiones del sistema que requieran el acceso al bus de direcciones o al bus de datos del ordenador. Es necesario tomar precauciones cuando se usa una expansión de este tipo, ya que es posible dañar el Commodore 64 por una mal función del equipo conectado.
El bus de expansión está dispuesto de la siguiente forma:



Las señales disponibles en el conector son las siguientes:

NOMBRE	C*	DESCRIPCION
GND	1	Tierra
+5VDC	2	(El total de los periféricos del port de usuario y
+5VDC	3	cartuchos no pueden exceder de 450mA.)
IRQ	4	Línea de petición de interrupción al 6502 (activa baja)
R/W	5	Lectura/Escritura
DOT CLOCK	6	Reloj para video de 8.18 MHz.
I/01	7	Bloque 1 E/S @$DE00-S$DEFF (activo bajo) sin buffer
GAME	8	Entrada activo bajo ttl
EXROM	9	Entrada activo bajo ttl
I/02	10	Bloque 2 E/S ↓$DF00-$DFFF (activo bajo) con buffer de salida ttl

*C=NUMERO DE CONTACTO

NOMBRE	C*	DESCRIPCION
ROML	11	Bloque de 8K. RAM/ROM decodificado @$8000 (activo bajo) con buffer ttl de salida
BA	12	Señal disponible del bus desde el chip VIC-II sin buffer 1 carga máx.
DMA	13	Línea de petición de acceso directo a memoria (entrada activo bajo) entrada ttl
D7	14	Bit del bus de datos 7
D6	15	Bit del bus de datos 6
D5	16	Bit de bus de datos 5
D4	17	Bit de bus de datos 4 sin buffer, 1 ttl c.máx
D3	18	Bit de bus de datos 3
D2	19	Bit de bus de datos 2
D1	20	Bit de bus de datos 1
D0	21	Bit de bus de datos 0
GND	22	Tierra
GND	A	Tierra
ROMH	B	8K. RAM/ROM decodificada @$E000 con buffer.
RESET	C	RESET 6502 (activo bajo) con buffer ttl de salida sin buffer de entrada
NMI	D	6502 Interrupción No Enmascarable (activo bajo) con buffer ttl de salida, sin buffer de entrada.
02	E	Reloj del sitema en fase 2
A15	F	Bit del bus de direcciones 15
A14	H	Bit del bus de direcciones 14
A13	J	Bit del bus de direcciones 13
A12	K	Bit del bus de direcciones 12
A11	L	Bit del bus de direcciones 11
A10	M	Bit del bus de direcciones 10
A9	N	Bit del bus de direcciones 9
A8	P	Bit del bus de direcciones 8 sin buffer, 1 ttl carga máxima
A7	R	Bit del bus de direcciones 7
A6	S	Bit del bus de direcciones 6
A5	T	Bit del bus de direcciones 5
A4	U	Bit del bus de direcciones 4
A3	V	Bit del bus de direcciones 3
A2	W	Bit del bus de direcciones 2
A1	X	Bit del bus de direcciones 1
A0	Y	Bit del bus de direcciones 0
GND	Z	Tierra

*C=NUMERO DE CONTACTO

A continuación se muestra una descripción de las líneas más importantes del port de expansión:

GND está conectado a la tierra del sistema.
DOT CLOCK Es el RELOJ, A 8.18 MHz. Todos los tiempos del sistema se derivan de este reloj.
BA es la señal del VIC–II sobre disponibilidad del bus. Esta línea pasa a bajo 3 ciclos antes de que el VIC–II actúe sobre los buses del sistema, y permanece baja hasta que el VIC–II ha terminado la tarea de mostrar nueva información.

DMA es la línea de ACCESO DIRECTO A MEMORIA. Cuando esta línea está baja, el bus de direcciones, el bus de datos y la línea de lectura/escritura del procesador 6510 entran en estado de alta impedancia. Esto significa que un procesador externo puede tomar el control de los buses del sistema. Esta línea sólo puede ser activada cuando el reloj 02 está bajo. También, puesto que el VIC–II sigue con sus tareas, el periférico externo debe adaptarse al ritmo del VIC–II. (Vea el diagrama del VIC–II). Esta línea está arriba en el Commodore 64.

CARTUCHO PROCESADOR Z–80

Leyendo este libro y practicando con su ordenador se habrá dado cuenta de lo versátil que es el Commodore 64. Para aumentar esta versatilidad y lograr que el ordenador sea aún más capaz, se han diseñado una serie de periféricos tales como el Datassette, la unidad de disco, la impresora...Todos estos aparatos pueden ser conectados a su Commodore 64 mediante los diversos ports de que dispone. Lo que hace que nuestros periféricos sean realmente buenos es que éstos son "inteligentes". Esto significa que no ocupan memoria RAM, siempre necesaria Usted dispone de los 64K. de memoria libres en su Commodore 64.
Otra ventaja de su Commodore 64 es que de hecho la mayoría de programas que escriba para el hoy serán compatibles con los ordenadores Commodore del mañana. Esto es posible gracias a la calidad del sistema operativo (OS) de nuestros ordenadores.
Sin embargo, hay una cosa que el sistema operativo del Commodore 64 no puede hacer lograr la compatibilidad de sus programas con los de un ordenador fabricado por otra compañía.
Probablemente usted no se habrá preocupado acerca del uso de otros ordenadores porque el Commodore 64 es muy fácil de usar. Sin embargo, para el usuario que desee disponer de software que no esté disponible en el formato de su Commodore 64 hemos creado el cartucho Commodore CP/M (R)
CP/M (R) no es un sistema operativo "dependiente del ordenador". Este sistema usa parte de la memoria disponible para programas para ejecutar su propio sistema operativo. Hay ventajas e inconvenientes en esto. Los inconvenientes son que los programas que escriba deben ser más cortos que los que podría escribir usando el sistema operativo de su Commodore 64. Además, usted NO puede usar las características del poderoso editor de pantalla de su Commodore. Las ventajas son que usted puede disponer de una vasta biblioteca de programas diseñados para trabajar con CP/M (R) y el microprocesador Z–80, y los programas que escriba en este sistema pueden ser transportados y ejecutados en otros ordenadores que dispongan del CP/M (R) y un procesador Z–80.
[illegible] ordenadores que disponen en opción de una tarjeta Z–80 requieren mani-

pulaciones en el interior del ordenador para instalarla, lo que puede estropear algún componente si no se tiene mucho cuidado. El cartucho Commodore CP/M (R) elimina este riesgo puesto que se conecta en el port de expansión sin que deba quitar ningún tornillo ni tocar ningún componente que más tarde podría causar problemas.

USO DEL COMMODORE CP/M (R)

El cartucho Commodore Z–80 le permite utilizar programas diseñados para funcionar con el microprocesador Z–80 en su Commodore 64. El cartucho se suministra con un diskette que contiene el sistema operativo CP/M (R).

EJECUCION DEL COMMODORE CP/M (R)

Para entrar en el sistema CP/M (R):

1) LOAD el programa CP/M desde su unidad de disco
2) Escriba RUN
3) Pulse la tecla **RETURN**

En este punto los 64K. de memoria RAM están disponibles para el microprocesador central 6510, o bien 48K. de RAM están disponibles para el procesador central Z–80. Usted puede pasar alternativamente de uno a otro procesador, pero NO puede usar los dos al mismo tiempo en un solo programa. El paso de un procesador a otro es posible gracias al sofisticado mecanismo de tiempo de su Commodore 64.

Puesto que las direcciones de memoria cambian al utilizar el cartucho Z-80, debe recordar que se debe añadir 4096 a las direcciones con las que trabaja en CP/M (R) para obtener las direcciones equivalentes en el sistema operativo normal de su Commodore 64. Las correspondencias de memoria entre los dos procesadores se muestran en la siguiente tabla:

Z-80 DIRECCIONES		6510 DIRECCIONES	
DECIMAL	HEX	DECIMAL	HEX
0000-4095	0000-0FFF	4096-8191	1000-1FFF
4096-8191	1000-1FFF	8192-12287	2000-2FFF
8192-12287	2000-2FFF	12288-16383	3000-3FFF
12288-16383	3000-3FFF	16384-20479	4000-4FFF
16384-20479	4000-4FFF	20480-24575	5000-5FFF
20480-24575	5000-5FFF	24576-28671	6000-6FFF
24576-28671	6000-6FFF	28672-32767	7000-7FFF
28672-32767	7000-7FFF	32768-36863	8000-BFFF
32768-36863	8000-8FFF	36864-40959	9000-9FFF
36864-40959	9000-9FFF	40960-45055	A000-AFFF
40960-45055	A000-AFFF	45056-49151	B000-BFFF
45056-49151	B000-BFFF	49152-53247	C000-CFFF
49152-53247	C000-CFFF	53248-57343	D000-DFFF
53248-57343	D000-DFFF	57344-61439	E000-EFFF
57344-61439	E000-EFFF	61440-65535	F000-FFFF
61440-65535	F000-FFFF	0000-4095	0000-0FFF

Para ACTIVAR el procesador Z-80 y DESACTIVAR el procesador 6510, escriba el siguiente programa:

```
10 REM ESTE PROGRAMA ES PARA SER USADO CON EL CARTUCHO DEL Z80
20 REM PRIMERO ALMACENA LOS DATOS DEL Z80 EN $1000(Z80=$0000)
30 REM ENTONCES DESCONECTA EL IRQ DEL 6510 Y FACILITA
40 REM EL CARTUCHO DEL Z80.EL CARTUCHO DEL Z80 DEBE SER DESCONECTADO
50 REM PARA FACILITAR EL SYSTEMA 6510
100 REM ALMACENA LOS DATOS DEL Z80
110 READ B:REM COGE EL TAMANO DEL CODIGO DEL Z80 QUE TIENE QUE SER MOVIDO
120 FOR I=4096 TO 4096+B-1:REM MUEVE EL CODIGO
130 READ A:POKE I,A
140 NEXT I
200 REM EJECUTA EL CODIGO DEL Z80
210 POKE 56333,127:REM DESCONECTA EL IRQ DEL 6510
220 POKE 56832,00:REM CONECTA EL CARTUCHO DEL Z80
230 POKE 56333,129:REM CONECTA EL IRQ DEL 6510 CUANDO HA SIDO HECHO EL
    CARTUCHO DEL Z80
240 END
1000 REM SECCION DE DATOS DEL CODIGO MAQUINA DEL Z80
1010 DATA 18:REM TAMANO DE LOS DATOS QUE DEBEN SER PASADOS
1100 REM CONECTA EL CODIGO DEL Z80
1110 DATA 00,00,00:REM NUESTRO CARTUCHO Z80 NECESITA CONECTAR TIEMPO EN $0000
1200 REM DATOS DE TRABAJO DEL Z80
1210 DATA 33,02,245:REM LD HL,NN(LOCALIZACION EN LA PANTALLA)
1220 DATA 52:REM INC HL(INCREMENTA ESA LOCALIZACION)
1300 REM DATOS DEL AUTO DESCONECTOR  DEL Z80
1310 DATA 62,01:REM LD A,N
1320 DATA 500,00,206:REM LD (NN),A:LOCALIZACION DEL I/O
1330 DATA 00,00,00,:REM NOP:NOP:NOP
1340 DATA 195,00,00:REM JMP $0000
```

Para más detalles acerca del CP/M (R) de Commodore y el cartucho procesador Z-80 lea el Manual DE Referencia del Z-80.

Apéndices

APENDICE A:

ABREVIATURAS DE LAS PALABRAS CLAVE DE BASIC:

Para ayudarle a ganar tiempo a la hora de mecanografiar los programas y comandos, el BASIC del Commodore-64 le permite abreviar la mayoría de las palabras claves. La abreviatura para "PRINT" es un interrogante. Las abreviaturas para otras palabras se consiguen escribiendo la primera o las dos primeras letras de la palabra, con SHIFT (o sea pulsando simultáneamente SHIFT y la letra). Si estas abreviaturas se utilizan en un programa, cuando sea listado aparecerá la palabra BASIC entera (sin abreviar). Observe que algunas de las palabras, cuando están abreviadas, incluyen un paréntesis izquierdo.

Palabra	Abreviatura	Símbolo en la pantalla	Palabra	Abreviatura	Símbolo en la pantalla
ABS	A SHIFT B	A	END	E SHIFT N	E
AND	A SHIFT N	A	EXP	E SHIFT X	E ♣
ASC	A SHIFT S	A ♥	FN	NINGUNA	FN
ATN	A SHIFT T	A	FOR	F SHIFT O	F
CHR$	C SHIFT H	C	FRE	F SHIFT R	F
CLOSE	CL SHIFT O	CL	GET	G SHIFT E	G
CLR	C SHIFT L	C	GET#	NINGUNA	GET#
CMD	C SHIFT M	C	GOSUB	GO SHIFT S	GO ♥
CONT	C SHIFT O	C	GOTO	G SHIFT O	G
COS	NINGUNA	COS	IF	NINGUNA	IF
DATA	D SHIFT A	D ♠	INPUT	NINGUNA	INPUT
DEF	D SHIFT E	D	INPUT#	I SHIFT N	I
DIM	D SHIFT I	D	INT	NINGUNA	INT

Palabra	Abreviatura	Símbolo en la pantalla	Palabra	Abreviatura	Símbolo en la pantalla
LEFT$	LE SHIFT F	LE	RIGHT$	R SHIFT I	R
LEN	NINGUNA	LEN	RND	R SHIFT N	R
LET	L SHIFT E	L	RUN	R SHIFT U	R
LIST	L SHIFT I	L	SAVE	S SHIFT A	S ♠
LOAD	L SHIFT O	L	SGN	S SHIFT G	S
LOG	NINGUNA	LOG	SIN	S SHIFT I	S
MID$	M SHIFT I	M	SPC(	S SHIFT P	S
NEW	NONE	NEW	SQR	S SHIFT Q	S ●
NEXT	N SHIFT E	N	STATUS	ST	ST
NOT	N SHIFT O	N	STEP	ST SHIFT E	ST
ON	NINGUNA	ON	STOP	S SHIFT T	S
OPEN	O SHIFT P	O	STR$	ST SHIFT R	ST
OR	NONE	OR	SYS	S SHIFT Y	S
PEEK	P SHIFT E	P	TAB(	T SHIFT A	T ♠
POKE	P SHIFT O	P	TAN	NINGUNA	TAN
POS	NINGUNA	POS	THEN	T SHIFT H	T
PRINT	?	?	TIME	TI	TI
PRINT#	P SHIFT R	P	TIME$	TI$	TI$
READ	R SHIFT E	R	USR	U SHIFT S	U ♥
REM	NINGUNA	REM	VAL	V SHIFT A	V ♠
RESTORE	RE SHIFT S	RE ♥	VERIFY	V SHIFT E	V
RETURN	RE SHIFT T	RE	WAIT	W SHIFT A	W ♠

CODIGOS DE PANTALLA

La siguiente tabla, contiene todos los caracteres que posee el Commodore 64 en su juego de caracteres. Muestra que número deberá entrar por POKE en la memoria de pantalla (registro 1024 a 2023) para obtener el carácter deseado. También se puede ver, a qué carácter le corresponde el número que se ha entrado por PEEK.
Hay dos juegos de caracteres disponibles pero sólo uno a la vez, esto quiere decir que no puede tener simultáneamente un pantalla caracteres de uno y otro juego. Se puede cambiar de juego pulsando simultáneamente las teclas SHIFT y COMMODORE.
Desde BASIC, haga POKE 53272,21 para obtener las mayúsculas y POKE 53272,23 para cambiar a minúsculas.
Cualquier carácter de la tabla puede ser presentado en modo INVERSO.
El código del carácter en INVERSO se obtiene, añadiendo al código del carácter "128".
Si quiere visualizar un círculo en la posición 1504, haga un POKE con el código del círculo (81) en el registro 1504: POKE 1504,81.
Hay también una posición de memoria correspondiente para el control del color de cada carácter presentado en pantalla (registros 55296-56295). Para cambiar el color del círculo a amarillo (código de color 7), hará un POKE en la posición de memoria correspondiente (55776) con el código: POKE 55776,7.
Refiérase al apéndice D para los mapas de memoria de pantalla y color, con sus códigos correspondientes.

CODIGOS DE PANTALLA

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
@		0	I	i	9	R	r	18
A	a	1	J	j	10	S	s	19
B	b	2	K	k	11	T	t	20
C	c	3	L	l	12	U	u	21
D	d	4	M	m	13	V	v	22
E	e	5	N	n	14	W	w	23
F	f	6	O	o	15	X	x	24
G	g	7	P	p	16	Y	y	25
H	h	8	Q	q	17	Z	z	26

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
[		27	9		57		W	87
£		28	:		58		X	88
]		29	;		59		Y	89
↑		30	<		60		Z	90
←		31	=		61			91
SPACE		32	>		62			92
!		33	?		63			93
"		34			64			94
#		35		A	65			95
$		36		B	66	SPACE		96
%		37		C	67			97
&		38		D	68			98
'		39		E	69			99
(		40		F	70			100
)		41		G	71			101
*		42		H	72			102
+		43		I	73			103
,		44		J	74			104
-		45		K	75			105
.		46		L	76			106
/		47		M	77			107
0		48		N	78			108
1		49		O	79			109
2		50		P	80			110
3		51		Q	81			111
4		52		R	82			112
5		53		S	83			113
6		54		T	84			114
7		55		U	85			115
8		56		V	86			116

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
		117			121			125
		118		☑	122			126
		119			123			127
		120			124			

Los códigos 128-125 son los mismos que los 0- 127 pero en video inverso.

APENDICE C

CODIGOS ASCII Y CHR$

Este Apéndice le muestra qué caracteres aparecen si escribe: PRINT CHR$(X), para todos los valores posibles de X. También le muestra los valores que obtendrá, si escribe PRINT ASC ("X"), donde x es cualquier carácter que pueda escribir. Esto es muy práctico para comparar un carácter recibido por una instrucción GET, para convertir la presentación de Mayúsculas a Minúsculas y escribir comandos (basados en caracteres) como el conmutador Mayúsculas/Minúsculas) que no podrían ser escritos dentro de las comillas de un PRINT.

Los códigos 192-223 son los mismos que los 96-127. Los códigos 224-254 son los mismos que los 160-190. El código 255 es el mismo que el 126.

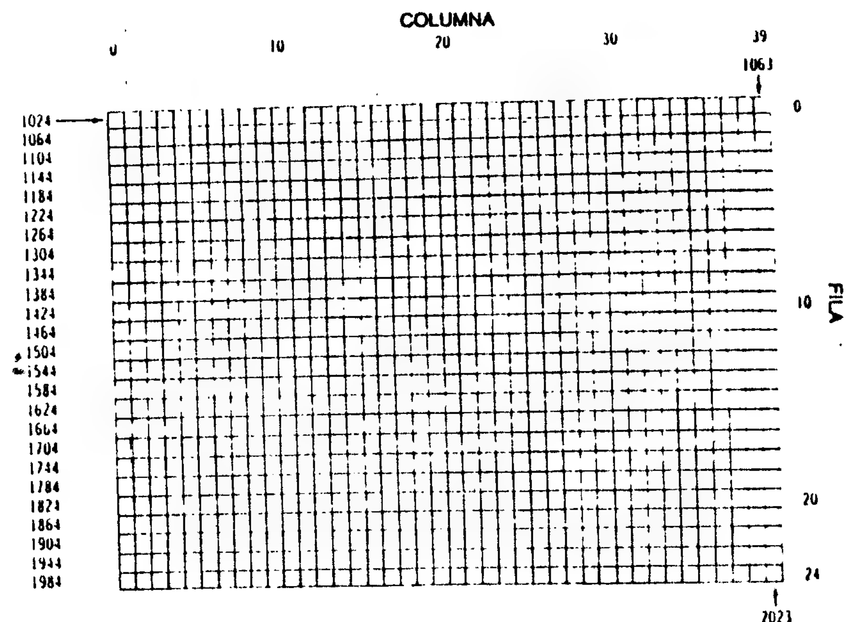
VISUALIZAR	CHR$	VISUALIZAR	CHR$	VISUALIZAR	CHR$	VISUALIZAR	CHR$
	0	CLR HOME	19	&	38	9	57
	1	INST DEL	20	'	39	:	58
	2		21	(	40	;	59
	3		22	)	41	<	60
	4		23	*	42	=	61
WHT	5		24	+	43	>	62
	6		25	,	44	?	63
	7		26	–	45	@	64
DISABLES SHIFT C=	8		27	.	46	A	65
ENABLES SHIFT C=	9	RED	28	/	47	B	66
	10	CRSR	29	0	48	C	67
	11	GRN	30	1	49	D	68
	12	BLU	31	2	50	E	69
RETURN	13	SPACE	32	3	51	F	70
SWITCH TO LOWER CASE	14	!	33	4	52	G	71
	15	"	34	5	53	H	72
	16	#	35	6	54	I	73
CRSR	17	$	36	7	55	J	74
RVS ON	18	%	37	8	56	K	75

VISUALIZAR	CHR$	VISUALIZAR	CHR$	VISUALIZAR	CHR$	VISUALIZAR	CHR$
L	76		105	f3	134		163
M	77		106	f5	135		164
N	78		107	f7	136		165
O	79		108	f2	137		166
P	80		109	f4	138		167
Q	81		110	f6	139		168
R	82		111	f8	140		169
S	83		112	SHIFT RETURN	141		170
T	84		113	SWITCH TO UPPER CASE	142		171
U	85		114		143		172
V	86		115		144		173
W	87		116		145		174
X	88		117	RVS OFF	146		175
Y	89		118	CLR HOME	147		176
Z	90		119	INST DEL	148		177
[	91		120		149		178
£	92		121		150		179
]	93		122		151		180
↑	94		123		152		181
←	95		124		153		182
	96		125		154		183
	97		126		155		184
	98		127	PUR	156		185
	99		128		157		186
	100		129		158		187
	101		130		159		188
	102		131	SPACE	160		189
	103		132		161		190
	104	f1	133		'62		191

LOS MAPAS DE MEMORIA DE PANTALLA COLOR Y PANTALA

Las tablas siguientes muestran, que situación de memoria controla el emplazamiento de los caracteres en la pantalla, y el registro que se utiliza para cambiar individualmente el color de un carácter , así como la lista de los códigos de colores.

MAPA DE LA MEMORIA DE PANTALLA

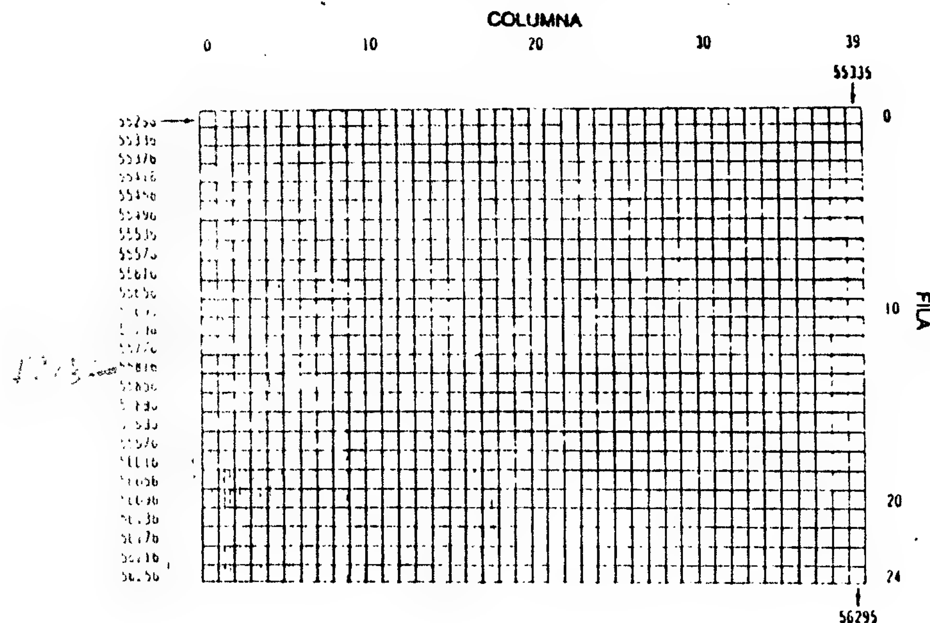


Los valores de los códigos de color que se pueden entrar por POKE, en un registro para cambiar el color de un carácter son los siguientes.

0 NEGRO	4 PURPURA	8 NARANJA	12 GRIS 2
1 BLANCO	5 VERDE	9 MARRON	13 VERDE claro
2 ROJO	6 Azul	10 ROJO claro	14 AZUL claro
3 CIAN	7 AMARIILO	11 GRIS 1	15 GRIS 3

Por ejemplo para cambiar el color de un carácter situado en la esquina izquierda superior de la pantalla, al rojo; escriba: POKE 55296,2.

MAPA DE LA MEMORIA DE COLORES



APENDICE E

VALORES DE LAS NOTAS MUSICALES

Este Apéndice contiene una lista completa de Notas musicales con el número de nota, el nombre de la nota, y los códigos a entrar por POKE en los registros de Alta y Baja Frecuencia del chip de sonidos para generar la nota deseada.

N.º de Nota	Nota Octava	Alta Frecuencia	Baja Frecuencia
0	C-0	1	18
1	C#-0	1	35
2	D-0	1	52
3	D#-0	1	70
4	E-0	1	90
5	F-0	1	110
6	F#-0	1	132
7	G-0	1	155
8	G#-0	1	179
9	A-0	1	205
10	A#-0	1	233
11	B-0	2	6
12	C-1	2	37
13	C#-1	2	69
14	D-1	2	104
15	D#-1	2	140
16	E-1	2	179
17	F-1	2	220
18	F#-1	3	8
19	G-1	3	54
20	G#-1	3	103
21	A-1	3	155
22	A#-1	3	210
23	B-1	4	12
24	C-2	4	73
25	C#-2	4	139
26	D-2	4	208
27	D#-2	5	25
28	E-2	5	103
29	F-2	5	185
30	F#-2	6	16
31	G-2	6	108
32	G#-2	6	206

N.º de Nota	Nota Octava	Alta Frecuencia	Baja Frecuencia
33	A–2	7	53
34	A#–2	7	163
35	B–2	8	23
36	C–3	8	147
37	C#–3	9	21
38	D–3	9	159
39	D#–3	10	60
40	E–3	10	205
41	F–3	11	114
42	F#–3	12	32
43	G–3	12	216
44	G#–3	13	156
45	A–3	14	107
46	A# -3	15	70
47	B–3	16	47
48	C–4	17	37
49	C#–4	18	42
50	D–4	19	63
51	D#–4	20	100
52	E–4	21	154
53	F–4	22	227
54	F#–4	24	63
55	G–4	25	177
56	G#–4	27	56
57	A–4	28	214
58	A#–4	30	141
59	B–4	32	94
60	C–5	34	75
61	C#–5	36	85
62	D–5	38	126
63	D#–5	40	200
64	E–5	43	52
65	F–5	45	198
66	F#–5	[illegible]	127
67	G–5	51	97
68	G#–5	54	111
69	A–5	57	172
70	A#–5	61	126
71	B–5	64	188
72	C–6	68	149
73	C#–6	72	[illegible]
74	D–6	76	252
75	D# -6	81	161
76	E–6	86	105
77	F6	91	140
78	F#–6	96	254
79	G–6	102	194
80	G#–6	108	223

N.º de Nota	Nota Octava	Alta Frecuencia	Baja Frecuencia
81	A–6	115	88
82	A#–6	122	52
83	B–6	129	120
84	C–7	137	43
85	C#–7	145	83
86	D–7	153	247
87	D# -7	163	31
88	E–7	172	210
89	F–7	183	25
90	F# -7	193	252
91	G–7	205	133
92	G#–7	217	189
93	A–7	230	176
94	A#–7	244	103

APENDICE F:

BIBLIOGRAFIA

Addison-Wesley	"BASIC and the Personal Computer", Dwyer and Critchfield
…ompute	"Compute's First Book of PET/CBM"
…wbay Computing	"Feed Me, I'm Your PET Computer", Carol Alexander
	"Looking Good with Your PET", Carol Alexander
	"Teacher's PET — Plans, Quizzes, and Answers"
…ative Computing	"Getting Acquainted With Your VIC 20", T. Hartnell
…ilhium Press	"BASIC Basic-English Dictionary for the PET", Larry Noonan
	"PET BA…", Tom Rugg and Phil Feldman
Faulk Baker Associates	"MOS Programming Manual", MOS Technology
Hayden Book Co.	"BASIC From the Ground Up", David E. Simon
	"I Speak BASIC to My PET", Aubrey Jones, Jr.
	"Library of PET Subroutines", Nick Hampshire
	"PET Graphics", Nick Hampshire
	"BASIC Conversions Handbook, Apple, TRS-80, and PET", David A. Brain, Phillip R. Oviatt, Paul J. Paquin, and Chandler P. Stone
Howard W. Sams	"The Howard W. Sams Crash Course in Microcomputers", Louis E. Frenzel, Jr.
	"Mostly BASIC: Applications for Your PET", Howard Berenbon
	"PET Interfacing", James M. Downey and Steven M. Rogers
	"VIC 20 Programmer's Reference Guide", A. Finkel, P. Higginbottom, N. Harris, and M. Tomczyk
Little, Brown & Co.	"Computer Games for Businesses, Schools, and Homes", J. Victor Nagigian, and William S. Hodges
	"The Computer Tutor: Learning Activities for Homes and Schools", Gary W. Orwig, University of Central Florida, and William S. Hodges
McGraw-Hill	"Hands-On BASIC With a PET", Herbert D. Peckman
	"Home and Office Use of VisiCalc", D. Castlewitz; and L. Chisauki
Osborne/McGraw-Hill	"PET/CBM Personal Computer Guide", Carroll S. Donahue
	"PET Fun and Games", R. Jeffries and G. Fisher
	"PET and the IEEE", A. Osborne and C. Donahue
	"Some Common BASIC Programs for the PET", L. Poole, M. Borchers, and C. Donahue
	"Osborne CP/M User Guide", Thom Hogan
	"CBM Professional Computer Guide"
	"The PET Personal Guide"
	"The 8086 Book", Russell Rector and George Alexy
P. C. Publications	"Beginning Self Teaching Computer Lessons"

Prentice-Hall	"The PET Personal Computer for Beginners", S. Dunn and V. Morgan
Reston Publishing Co.	"PET and the IEEE 488 Bus (GPIB)", Eugene Fisher and C. W. Jensen
	"PET BASIC—Training Your PET Computer", Ramon Zamora, Wm. F. Carrie, and B. Allbrecht
	"PET Games and Recreation", M. Ogelsby, L. Lindsey, and D. Kunkin
	"PET BASIC", Richard Huskell
	"VIC Games and Recreation"
Telmas Courseware Ratings	"BASIC and the Personal Computer", T. A. Dwyer, and M. Critchfield
Total Information Services	"Understanding Your PET/CBM, Vol. 1, BASIC Programming"
	"Understanding Your VIC", David Schultz

APENDICE G:

MAPA DE LOS REGISTROS DEL VIC CHIP

Registro N.º Decim.	Hexag	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
0	0	S0X7							S0X0	Componente X SPRITE 0
1	1	S0Y7							S0Y0	Componente Y SPRITE 0
2	2	S1X7							S1X0	SPRITE 1X
3	3	S1Y7							S1Y0	SPRITE 1Y
4	4	S2X7							S2X0	SPRITE 2X
5	5	S2Y7							S2Y0	SPRITE 2Y
6	6	S3X7							S3X0	SPRITE 3X
7	7	S3Y7							S3Y0	SPRITE 3Y
8	8	S4X7							S4X0	SPRITE 4X
9	9	S4Y7							S4Y0	SPRITE 4Y
10	A	S5X7							S5X0	SPRITE 5X
11	B	S5Y7							S5Y0	SPRITE 5Y
12	C	S6X7							S6X0	SPRITE 6X
13	D	S6Y7							S6Y0	SPRITE 6Y
14	E	S7X7							S7X0	Componente X SPRITE 7
15	F	S7Y7							S7Y0	Componente Y SPRITE 7
16	10	S7X8	S6X8	S5X8	S4X8	S3X8	S2X8	S1X8	S0X8	MSB de Coordenad. X
17	11	RC8	EC5	BSM	BLNK	RSEL	YSCL2	YSCL1	YSCL0	Scroll/Modo
18	12	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	RASTREO
19	13	LPX7							LPX0	LAPIZ-OPT. X
20	14	LPY7							LPY0	LAPIZ-OPT. Y

Registro N.º Decm.	Hexag.	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
21	15	SE7							SE0	LANZAMIEN. SPRITE ON/OFF
22		N C	N.C.	RSI	MCM	CSEL	XSCL2	XSCL1	XSCL0	Scroll/Modo
23	17	SEXY7							SEXY0	ANULACION SPRITE Y
24	18	VS13	VS12	VS11	CB13	CB12	CB11	CB10	N C	Memoria de caracter de PANTALLA
25	19	IRQ	N C	N.C.	N.C.	LPIRQ	ISSC	ISBC	RIRQ	Interruptor DEMANDA
26	1A	N.C.	N.C	N.C.	N.C.	MLPI	MISSC	MISBC	MRIRQ	Interruptor DEMANDA MASCARAS?
27	1B	BSP7							BSP0	Prioridad fondo SPRITE
28	1C	SCM7							SCM0	SELECTOR COLOR SPRITE
29	1D	SEXX7							SEXX0	AMPLIACION SPRITE X
30	1E	SSC7							SSC0	CHOQUE DE SPRITES
31	1F	SBC7							SBC0	CHOQUE DE FONDO Y SPRITE

				CODIGOS DE COLOR		DEC.	HEXA.		COLOR
32	20	0	0	NEGRO	EXT 1				COLOR EXTERIOR
33	21	1	1	BLANCO	BKGD0				
34	22	2	2	ROJO	BKGD1				
35	23	3	3	TURQUESA	BKGD2				
36	24	4	4	MALVA/VIOLETA	BKGD3				
37	25	5	5	VERDE	SMC 0				SPRITE MULTICOLOR 0
38	26	6	6	AZUL	SMC 1				1
39	27	7	7	AMARILLO	S0COL				SPRITE COLOR 0
40	28	8	8	NARANJA	S1COL				1
41	29	9	9	MARRON	S2COL				2
42	2A	10	A	ROJO	S3COL				3
43	2B	11	B	GRIS 1	S4COL				4
44	2C	12	C	GRIS 2	S5COL				5
45	2D	13	D	VERDE CL.	S6COL				6
46	2E	14	E	AZUL CL.	S7COL				7
		15	F	GRIS 3					

NOTA:
SOLO LOS COLORES DE 0 A 7 PUEDEN SER UTILIZADOS EN MULTICOLOR.

APENDICE H:

DERIVACION DE FUNCIONES MATEMATICAS

Las funciones que no son intrínsecas al BASIC del Commodore 64 pueden calcularse de la forma siguiente:

FUNCION	EQUIVALENTE EN BASIC
SECANTE	SEC(X)=1/COS(X)
COSECANTE	CSC(X)=1/SIN(X)
COTANGENTE	COT(X)=1/TAN(X)
SENO INVERSO	ARCSIN(X)=ATN(X/SQR(-X*X+1))
COSENO INVERSO	ARCCOS(X)=ATN(X/SQR(-X*X+1))+π/2
SECANTE INVERSA	ARCSEC(X)=ATN(X/SQR(X*X-1))
COSECANTE INVERSA	ARCCSC(X)=ATN(X/SQR(X*X-1))+(SGN(X)-1*π/2
COTANGENTE INVERSA	ARCOT(X)=ATN(X)+π/2
SENO HIPERBOLICO	SINH(X)=(EXP(X)-EXP(-X))/2
COSENO HIPERBOLICO	TANH(X)=EXP(-X)/(EXP(x)+EXP(-X))*2+1
SECANTE HIPERBOLICA	SECH(X)=2/(EXP(X)+EXP(-X))
COSECANTE HIPERBOLICA	CSCH(X)=2/(EXP(X)-EXP(-X))
COTANGENTE HIPERBOLICA	COTH(X)=EXP(-X)/(EXP(X))-EXP(-X))*2+1
SENO HIPERBOLICO	ARCSIN(X)=LOG(X+SQR(X*X+1))
COSENO HIPERBOLICO INVERSO	ARCCOSH(X)=LOG(X+SQR(X*X-1))
TANGENTE HIPERBOLICA INVERSA	ARCTANCH(X)=LOG((1+X)/(1-X))/2
SECANTE HIPERBOLICO INVERSO	ARCSECH(X)=LOG((SQR(-X*X+1)+1/X)
COSECANTE HIPERBOLICA INVERSA	ARCCSCH(X)=LOG((SGN(X)*SQR(X*X+1/x)
COTANGENTE HIPERBOLICA INVERSA	ARCCOTH(X)=LOG((X+1)/(x-1))/2

APENDICE I:

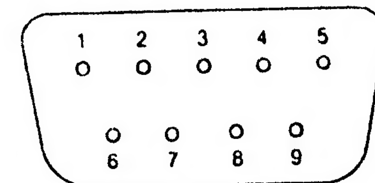
CONEXIONES DE DISPOSITIVO DE ENTRADA/SALIDA

Este apéndice está pensado para mostrarle que conexiones se pueden hacer con el Commodore 64.

1) E/S Juegos
2) Conexión Cart.
3) Audio-Video
4) E/S Serie (Disco/IMP.)
5) Salida Modulador
6) Cassete
7) Port del usuario

Control del Port 1

Contacto Púa n.º	TIPO	NOTA
1	JOY A0	
2	JOY A1	
3	JOY A2	
4	JOY A3	
5	Potencióm. A4	
6	botón A láp. óp.	
7	+5V	Máx. 50 mA
8	TIERRA	
9	Potencióm. AX	



1) E/S Juegos

Control del Port 2

Contacto Púa n.º	TIPO	NOTA
1	JOY B0	
2	JOY B1	
3	JOY B2	
4	JOY B3	
5	Potencióm. B4	
6	botón B láp. óp.	
7	+5V	Máx. 50 mA
8	TIERRA	
9	Potencióm. BX	

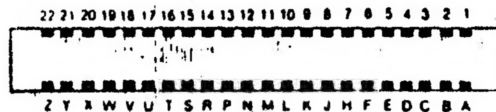
Expansión para cartuchos:

Contacto	Tipo
12	BA
13	DMA
14	D7
15	D6
16	D5
17	D4
18	D3
19	D2
20	D1
21	D0
22	TIERRA

Contacto	Tipo
N	A9
P	A8
R	A7
S	A6
T	A5
U	A4
V	A3
W	A2
X	A1
Y	A0
Z	TIERRA

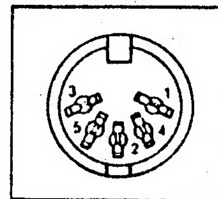
Contacto	Tipo
1	TIERRA
2	+5V
3	+5V
4	IRQ
5	R/W
6	Dot Clock
7	I/0 2
8	GAME
9	EXROM
10	I/0 2
11	ROML

Contacto	Tipo
A	GND
B	ROMH
C	RESET
D	NMI
E	S 02
F	A15
H	A14
J	A13
K	A12
L	A11
M	A10



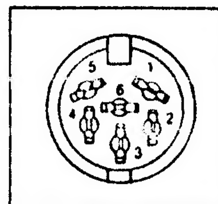
Audio/Video

Contacto	Tipo	Nota
1	LUMINANCIA	
2	TIERRA	
3	SALIDA AUDIO	
4	SALIDA VIDEO	
5	ENTRA AUDIO	



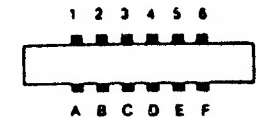
E/S de Serie (impresora/disco)

Contacto	Tipo
1	SERIAL SQRIN
2	TIERRA
3	SERIAL ANT. IN/OUT
4	SERIAL CLK IN/OUT
5	SERIAL DATA IN/OUT
6	RESET



El Cassette

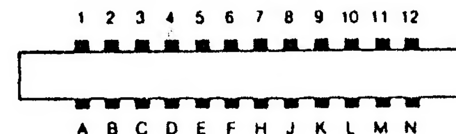
Contacto	Tipo
A-1	TIERRA
B-2	+5V
C-3	MOTOR DEL CASSETTE
D-4	LECTURA DEL CASSETTE
E-5	GRABACION DEL CASSETTE
F-6	INTERRUPTOR CASSETTE



El Port del Usuario:

Contacto	Tipo	Nota
1	TIERRA	Máx. 100 mA
2	+5V	
3	RESET	
4	CNT1	
5	SP1	
6	CNT2	
7	SP2	
8	PC2	
9	ENT. SERIAL ANT.	
10	9V≃	Máx. 100 mA
11	9V≃	Máx. 100 mA
12	TIERRA	

Contacto	Tipo	Nota
A	TIERRA	
B	FLAG2	
C	PB0	
D	PB1	
E	PB2	
F	PB3	
H	PB4	
J	PB5	
K	PB6	
L	PB7	
M	PA2	
N	TIERRA	



CONVERSION DE PROGRAMAS BASIC ESTANDARD A BASIC DEL COMMODORE 64

Si tiene programas escritos en otro BASIC que el de Commodore, será necesario hacer unos pequeños retoques, antes de ejecutarlos con el Commodore 64, hemos incluido algunos consejos para hacer la conversión más sencilla.

Dimensiones de Cadenas
Borre todas las instrucciones que declaran la longitud de la cadena. Una instrucción DIM A$(I,J) que dimensiona una matriz de cadenas para J elementos de longitud I, deberán ser convertidos a la sentencia de BASIC Commodore DIM A$ (J).
Algunos BASICs utilizan una coma o signos para la concatenacción de cadenas. Cada una de ellas debe ser cambiada por un signo "+" que es el operador del BASIC-Commodore para la concatación de cadenas (alfabéticas).
En el Basic del Commodore 64, las funciones MID$, RIGHT$: LEFT$, se utilizan para obtener subcadenas de cadenas. Formatos como A$(1) para acceder al carácter de orden I en A$, o A$(I,J) para sacar una subcadena de A$ de la posición I a J, deben cambiarse así:

Otros BASICs	BASIC del Commodore 64
A$(I) = X$	A$ = LEFT$(A$,I−1)+X$+MID$(A$,I+1)
A$(I,J) = X$	A$ = LEFT$(A$,I−1)+X$+MID$(A$,J+1)

Asignaciones Múltiples
Para poner a cero. B y C algunos BASICs le permiten instrucciones de este tipo:
1ØLET B = C = Ø

El BASIC del Commodore 64 interpretaría el segundo signo = como un operador lógico y ajustaría B = −1 si C = 0. Para evitar este problema convierte la instrucción en la siguiente:
1ØC = Ø:B = Ø'

Instrucciones Múltiples
Algunos BASICs utilizan una barra (atras) "" pra separar diversas instrucciones en una misma línea. Con el BASIC-Commodore separe todas las instrucciones con los 2 puntos (:)

Funciones MAT
Los programas que utilizan las instrucciones MAT algunos BASICs deben ser escritas utilizando un bucle FOR...NEXT para ser ejecutadas debidamente.

MENSAJE DE ERROR

Este apéndice contiene una lista completa de los mensajes de error generados por el Commodore 64 y la descripción de las causas:

BAD DATA...(Datos incorrectos): Se recibieron datos alfanúmericos desde un fichero abierto, cuando el programa esperaba datos númericos.
BAD SUBSCRIPT...(Subscrito incorrecto): El programa intentaba hacer referencia a un elemento de una matriz cuyo número estaba fuera de la gama especificada en la instrucción DIM.
CAN'T CONTINUE...(No es posible continuar): El comando CONT no puede operar porque no había empezado la ejecución del programa, hay un error o se ha cambiado una línea.
DEVICE NOT PRESENT...(Falta periférico): El periférico requerido no está disponible para una instrucción OPEN, CLOSE, CMD, PRINT#, INPUT# o GET#.
DIVISION BY ZERO...(División por cero): La división por cero no tiene sentido matemático y no está permitida.
EXTRA IGNORED...(Se omite los datos extra): Se mecanografiaron demasiados datos en respuesta a una instrucción INPUT. Se aceptaron sólo los primeros datos.
FILE NOT FUND...(Fichero no encontrado): Si usted buscaba un fichero en una cinta, se encontró una marca END OF TAPE (final de cinta). Si se buscaba en un diskette, no existe fichero con dicho nombre.
FILE NOT OPEN...(Fichero no abierto): El fichero especificado en una instrucción CLOSE, CMD, PRINT#, IMPUT#, o GET# debe abrirse primero con OPEN.
FILE OPEN...(Fichero abierto): Se ha intentado abrir un fichero usando el número de un fichero ya abierto.
FORMULA TOO COMPLEX...(Fórmula demasiado completa): la expresión de cadena evaluada debería dividirse por lo menos en dos partes para que el sistema pudiera elaborarla.
ILLEGAL DIRECT...(Ilegal directo): La instrucción INPUT sólo puede usarse incorporada en un programa, no en modo directo.
ILEGAL QUANTITY...(Cantidad ilegal): Un número usado como el argumento de una función o instrucción está fuera de la gama permitida.
LOAD: Hay un problema con el programa en la cinta.
NEXT WITHOUT FOR... (NEXT sin FOR): Hay bucles encajados incorrectamente o un nombre de variable en una instrucción NEXT que no corresponde a la de la instrucción FOR.
NOT INPUT FILE...Se han intentado introducir o extraer datos con INPUT o GET en un fichero especificado sólo para salida de datos.
NOT OUTPUT FILE...(No fichero de salida): Se ha intentado grabar datos con PRINT# en un fichero que estaba especificado sólo para la extracción de datos.
OUT OF DATA... (Datos agotados): Se ejecutó una instrucción READ, pero no quedaban datos por leer en la instrucción DATA.

OUT OF MEMORY...(Memoria agotada): No queda memoria RAM para el programa o sus variables. Puede ocurrir también cuando se encajan demasiados bucles FOR o hay demasiadas instrucciones GOSUB.
OVERFLOW... (No cabe): El resultado de un cálculo es mayor que el máximo número permitido, que es 1.7141884/ + 38.
REDIM'D ARRAY...Matriz redimensionada): Una matriz puede dimensionarse sólo una vez, con DIM. Si se usa una variable de matriz antes de que ésta sea dimensionada, se ejecuta una operación DIM automáticamente, estableciendo el número de elementos de dicha matriz en diez, y cualquier posterior dimensionado generará el mensaje de error.
REDO FROM START...(Volver a empezar desde el principio): Se mecanografiaron datos de caracteres durante una instrucción INPUT, en vez de los datos numéricos que se esperaban. Simplemente volver a efectuar la entrada en el teclado en modo que sea correcta y el programa continuará por si mismo.
RETURN WITHOUT GOSUB...(RETURN sin GOSUB): Se encontró una instrucción RETURN sin que se hubiera incorporado un comando GOSUB.
STRING TOO LONG...(Cadena demasiado larga): Una cadena puede contener como máximo 255 caracteres.
SYNTAX...(Sintanxis): Una instrucción no es reconocida por el VIC: hay un paréntesis de más o menos, un error de mecanografiado, etc.
TYPE MISMATCH...(Los datos mecanografiados no coinciden): Este mensaje de error se genera cuando se usa un número en vez de una cadena, o viceversa.
UNDEF'D FUNCTION...(Función no definida): se usó como referencia una función definida por el usuario, pero la misma no había sido definida usando la instrucción DEF FN.
UNDEF'D STATEMENT...(Instrucción no definida): Se intentó pasar con GOTO o GOSUB o ejecutar con RUN una línea que no existía.
VERIFY...(Verificar): El programa en la cinta cassette o en el disco no coincide con el programa actualmente en la memoria del ordenador.